Regras de associação

- Disciplina: Tópicos: Aprendizado de Máquina
- Profa. Dra. Adriane Beatriz de Souza Serapião
- Ciências da Computação – UNESP – Rio Claro

Introdução

- As bases de dados apresentadas até agora compartilham uma propriedade comum: todas elas são compostas por um conjunto de objetos caracterizados por um conjunto de características (atributos).
- Existe outro tipo de base de dados, muito comum em alguns ambientes empresariais, que relaciona conjuntos de itens que ocorrem em transações.
- Esses dados são muito valiosos para os negócios, pois permitem que sejam extraídas informações sobre o comportamento de compras de cada cliente, podendo ser usados na realização de promoções e campanhas de marketing, gestão de estoques, definição de catálogos, análise de perdas, relacionamento com clientes e muitas outras ações.

Introdução

- O exemplo típico é o do *carrinho de supermercado*.
- Quando alguém vai ao supermercado, faz uma compra e passa na caixa registradora, as informações de quais produtos foram comprados, a quantidade e os preços pagos ficam armazenados no banco de dados do supermercado. Cada registro desses é chamado de *transação* e, por isso, tais bases de dados são denominadas *transacionais*.

Tabela 7.1 Exemplo de base de dado transacional

TID	Itens
1	{Leite, pão, açúcar, café, manteiga}
2	{Mamão, banana, maçã}
3	{Leite, pão}
4	{Leite, pão, manteiga, banana}

Definição do problema

- Para que seja feita a mineração das regras de associação, as bases de dados transacionais normalmente são representadas seguindo o mesmo padrão das bases de dados convencionais, ou seja, com os objetos nas linhas e os atributos nas colunas.
- A diferença é que os atributos das bases transacionais são os itens que aparecem nas transações, o que faz com que tais bases de dados facilmente apresentem alta dimensionalidade, da ordem de centenas e até milhares de itens (atributos).

Tabela 7.2 Base de dados transacional da Tabela 7.1 representada como uma base binária

TID	Leite	Pão	Açúcar	Café	Manteiga	Mamão	Banana	Maçã
1	1	1	1	1	1	0	0	0
2	0	0	0	0	0	1	1	1
3	1	1	0	0	0	0	0	0
4	1	1	0	0	1	0	1	0

Definição do problema

- Dado um conjunto de transações, onde cada transação é composta por um conjunto de itens, uma regra de associação é uma regra $X \rightarrow Y$, na qual X e Y são conjuntos de itens.
- O significado intuitivo de uma regra de associação é que as transações em uma base de dados que contêm itens em X também contêm itens em Y.
- Assim, as regras de associação podem ser vistas como padrões descritivos que representam a probabilidade de que um conjunto de itens apareça em uma transação, dado que outro conjunto está presente.

Regras de associação

- **Mineração de associações ou de regras de associação:**
  - Encontrara padrões frequentes, associações, correlações, ou estruturas causais a partir de conjuntos de itens ou objetos em DB de transações, relacionais, ou em outros repositórios de informações.
- **Aplicações:**
  - Análise de cestas de dados (*basket data*), *marketing* cruzado, projeto de catálogos, agrupamento, etc.

Regras de associação

- **Exemplo**:
  - Uma base de dados de um supermercado teria como regra o fato de que 80% dos clientes que compram um produto Q, também adquirem, na mesma ocasião o produto W. Em que 80% é o fator de confiabilidade da regra.
- **Problema**:
  - Analisar um grande volume de conhecimento extraído no formato de regras.

Regras de associação
Conceitos importantes

Definição

Seja ***D*** uma Base de Dados composta por um conjunto de itens:

$$\boldsymbol{A = \{a_1, a_2, \ldots, a_m\}}$$

ordenados lexicograficamente e por um conjunto de transações:

$$\boldsymbol{T = \{t_1, t_2, .., t_n\}},$$

em que, cada transação $\boldsymbol{t_i}$ é composta por um conjunto de ítens tal que $\boldsymbol{t_i}$ está contido em ***A*****.**

Regras de associação
Conceitos importantes

- ***Itemset***

Conjunto de atributos ou itens ordenados lexicograficamente.

Exemplos:

{a, b ,c}

{1, 2 ,3}

{André, Marcio, Marcos}

Regras de associação
Conceitos importantes

Exemplo:

- Conjunto de itens:
  {produto1, produto2, produto3}

- Conjunto de transações T (compras de clientes):
  t_1: produto1, produto2
  t_2: produto1, produto2, produto3
  t_3: produto2, produto3

Regras de associação
Conceitos importantes

Uma regra de associação é uma implicação na forma:

$$\text{LHS} \longrightarrow \text{RHS}$$

em que LHS e RHS são conjuntos de itens que estão contidos em A e a intersecção de LHS e RHS é vazia.

Regras de associação
Conceitos importantes

Exemplo:

- ***Itemset***: {a b c}

- **Regras**:
  - a $\rightarrow$ bc
  - b $\rightarrow$ ac
  - a $\rightarrow$ b
  - a $\rightarrow$ c
  - c $\rightarrow$ ab
  - .....

Regras de associação
Conceitos importantes

- A regra LHS $\longrightarrow$ RHS ocorre no conjunto de transações T com confiança *c* se *c*% das transações em T em que ocorre LHS também ocorre RHS.

- A regra LHS $\longrightarrow$ RHS tem suporte *s* se em *s*% das transações em D ocorre LHS $\longrightarrow$ RHS.

Regras de associação
Conceitos importantes

- **Suporte:**
  - Indica a frequência com que um *itemset* ou com que LHS e RHS ocorrem juntos no conjunto de dados.

- **Exemplos:**
  - ***Itemset*:**

    {be} **Suporte = 1**

  - **Regra:**

    ab -> c = {abc} **Suporte = 2**

X1	X2	X3
a	b	c
a	b	e
a	b	c

Regras de associação
Conceitos importantes

- ***Itemset* frequente**

  É um *itemset* com suporte maior ou igual a um suporte mínimo especificado pelo usuário.

X1	X2	X3
a	b	c
a	b	e
a	b	c

Exemplo:

Suporte Mínimo = 2

{ab} ***Suporte = 3***

{ab} é um itemset frequente

Regras de associação
Conceitos importantes

- **Confiança**

Indica a frequência com que LHS e RHS ocorrem juntos em relação ao número total de registro em que LHS ocorre.

X1	X2	X3
a	b	c
a	b	e
a	b	c

Exemplo: **ab -> c**

$$\text{confiança} = \frac{\text{suporte}(\{LHS \wedge RHS\})}{\text{suporte}(LHS)}$$

$$\text{confiança} = \frac{\text{suporte}(\{abc\})}{\text{suporte}(\{ab\})} = \frac{2}{3} = 0.66$$

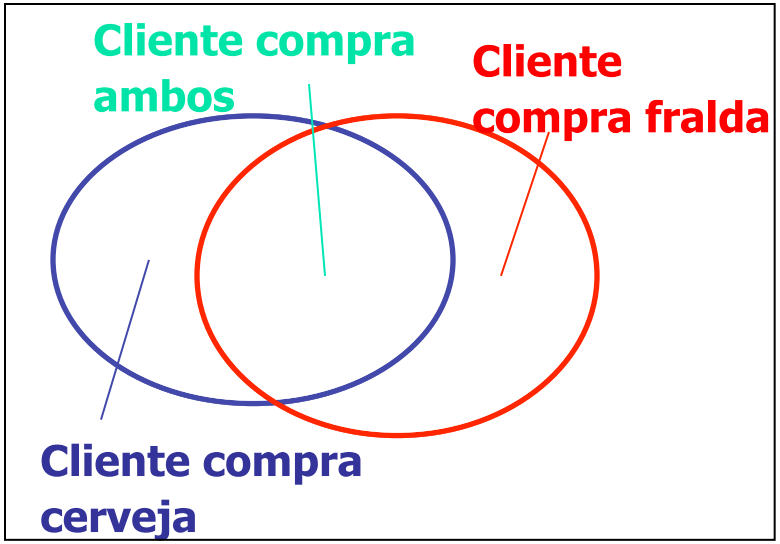
Regras de associação
Conceitos importantes

Transaction 1	🍎 🍺 🍚 🍗
Transaction 2	🍎 🍺 🍚
Transaction 3	🍎 🍺
Transaction 4	🍎 🍐
Transaction 5	🍼 🍺 🍚 🍗
Transaction 6	🍼 🍺 🍚
Transaction 7	🍼 🍺
Transaction 8	🍼 🍐

Support {🍎} = 4

Confidence {🍎 → 🍺} = Support {🍎, 🍺} / Support {🍎}

Regras de associação



Encontrar regras X & $Y \Rightarrow Z$ **com** suporte e confiança mínimos

- Suporte, s, é a probabilidade de uma transação conter $\{X \cap Y \cap Z\}$
- Confiança, c, é a probabilidade condicional da transação tendo $\{X \cap Y\}$ também conter Z

Transação	Itens
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Para um suporte mínimo de 50%, e confiança mínima de 50%, tem-se:

- $A \Rightarrow C$ (50%, 66,6%)
- $C \Rightarrow A$ (50%, 100%)

Regras de associação

- **Exemplo:**
  - Formato da regra:
    "corpo (LHS) => cabeça (RHS) [suporte, confiança]";
  - compra(X, "fraldas") => compra (X, "cerveja") [0,5%, 60%]

Exemplo

A = {bermuda, calça, camiseta, sandália, tênis} e
T = {1, 2, 3, 4}.

- Suporte Mínimo = 50% (2 transações).
- Confiança Mínima = 50%.

Transações	Itens Comprados
1	calça, camiseta, tênis
2	camiseta, tênis
3	bermuda, tênis
4	calça, sandália

***Itemsets* Frequentes**	**Suporte**
{tênis}	75%
{calça}	50%
{camiseta}	50%
{camiseta, tênis}	50%

Exemplo

- Suporte Mínimo = 50% (2 transações)
- Confiança Mínima = 50%

Itemsets **Frequentes**	**Suporte**
{tênis}	75%
{calça}	50%
{camiseta}	50%
{camiseta, tênis}	50%

tênis $\rightarrow$ *camiseta,* em que:

$$\text{suporte} = \text{suporte}(\{\text{tênis}, \text{camiseta}\}) = 50\%$$

$$\text{confiança} = \frac{\text{suporte}(\{\text{tênis, camiseta}\})}{\text{suporte}(\{\text{tênis}\})} = \frac{50}{75} = 66{,}6\%$$

Geração de regras de associação

- Esquema básico dos algoritmos:
  - Dado uma Base de Dados *D* composta por um conjunto de itens $A = \{a_1,a_2,...,a_m\}$ ordenados lexicograficamente e por um conjunto de transações $T=\{t_1,t_2,..,t_n\}$, em que, cada transação ti é composta por um conjunto de itens tal que t_i está contido em *A*.

Geração de regras de associação

- Encontrar todos os *itemsets* que possuem suporte maior que um suporte mínimo especificado pelo usuário (*itemsets* frequentes).

- Utilizar todos os *itemsets* frequentes para gerar todas as regras de associação que possuem confiança maior do que a confiança mínima especificada pelo usuário.

Regras de associação

- Algoritmo utilizado:
  - *APRIORI*.
  - Princípio: todo subconjunto de um *itemset* frequente deve ser frequente.
  - Várias otimizações para melhoria da performance computacional.

Algoritmo *Apriori*

- Proposto por Agrawal et al (1993).
- É um modelo estudado extensivamente pelas comunidades de bancos de dados e aprendizado de máquina.
- Assume que os dados são categóricos; portanto, não se aplica a dados numéricos.
- Inicialmente utilizado na análise de cesta de compras em supermercados (*Market Basket Analysis) para* determinar como os itens comprados por clientes estão relacionados:
  - {leite, pão} → {manteiga} [sup = 5%, conf = 100%]

Algoritmo *Apriori*

- Encontra todos os *k-itemsets* frequentes contidos em uma base de dados:
  - Gera um conjunto de *k-itemsets* candidatos e então percorre a base de dados para determinar se os mesmos são frequentes, identificando desse modo todos os *k-itemsets* frequentes.

Algoritmo *Apriori*

1. L_1 := {1-itemsets frequentes};

2. para (k := 2; $L_{k-1} \neq \varnothing$; k ++) faça

3. C_k := apriori-gen (L_{k-1}); //Gera novos conjuntos candidatos

4. para todo (transações $t \in T$) faça

5. C_t := subset (C_k, t); //Conjuntos candidatos contidos em t

6. para todo candidatos $c \in C_t$ faça

7. c.count ++;

8. fim-para

9. fim-para

10. L_k := {$c \in C_k$ | c.count $\leq$ sup-min };

11. fim-para

12. Resposta := $\bigcup_k L_k$

Exemplo

A = {bermuda, calça, camiseta, sandália, tênis}

1-itemsets: {bermuda}, {calça}, {camiseta}, {sandália}, {tênis}

L1 = {{tenis} , {calça}, {camiseta}}
(1-itemset frequente)

C2= {{tenis,bermuda}, {tenis,calça}, {tenis, camiseta}, ..., {calça, bermuda},...}
(2-itemsets candidatos)

L2= {{tenis, camiseta}}
(2-itemsets frequentes)

Algoritmo para gerar regras de associação

- Gera um conjunto de regras de associação a partir de um conjunto contendo todos *k-itemsets* frequentes, com $k \geq 2$.

Algoritmo para gerar regras de associação

Function apriori-gen(L)

1. para todo (k-itemset frequente l_k, k ≥ 2) faça

2. Call genrules (l_k, l_k);

3. fim-para

// O procedimento genrules gera todas as regras válidas sobre l_k

4. procedure genrules (l_k : k-itemset frequente, a_m: m-itemset frequente)

5. gerar subconjuntos não vazios de um itemset frequente com m-1 itens

6. para todo itemset gerado construir regras

7. se confiança da regra gerada ≥ confiança mínima

8. imprimir regra

9. se (m-1>1) Call genrules (l_k, a_{m-1}); //Gera regras com subconjuntos de a_{m-1} como antecedente

10. fim-se

11. fim-se

12. fim-para

Exemplo

Itemset: {camiseta, tenis}

genrule gera subconjuntos:

subconjunto a1 = {tenis}

subconjunto a2 = {camiseta}

constrói regras com esses *itemsets* no lado esquerdo e calcula a confiança:

regra gerada pelo subconjunto a1: **tenis -> camiseta**

$$\text{suporte} = \text{suporte}(\{\text{tênis}, \text{camiseta}\}) = 50\%$$

Exemplo

regra gerada pelo subconjunto a1:

tenis -> camiseta
suporte = suporte({tenis, camiseta}) = 50%
confiança = suporte({tenis, camiseta}) /
suporte({tenis}) = 50/75 = 66,66%

regra gerada pelo subconjunto a2:

camiseta -> tenis
suporte = suporte({camiseta, tenis}) = 50%
confiança = suporte({camiseta, tenis}) /
suporte({camiseta}) = 50/50 = 100%

Regras de associação – exemplo

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
{2 3 5}	2

Regras de associação

- Regras de associação multiníveis:
  - Pressupõe uma hierarquia;
  - Abordagem *top-down* progressiva;
  - Inicialmente: encontrar as regras "fortes" de alto nível:
    - Leite => pão [20%, 60%]
  - Em seguida, regras "fracas" de mais baixo nível:
    - 2% leite => pão branco [6%, 50%]

Não abordada neste momento.

Tarefa de Mineração

Dados :

- Um banco de dados de transações *D*
- Um limite de suporte N, $1 \geq N > 0$
- Um limite de confiança M, $1 \geq M > 0$

Problema:

Encontrar todas as regras de associação *r* em *D* tais que:

$$\text{Sup}(r) \geq N$$

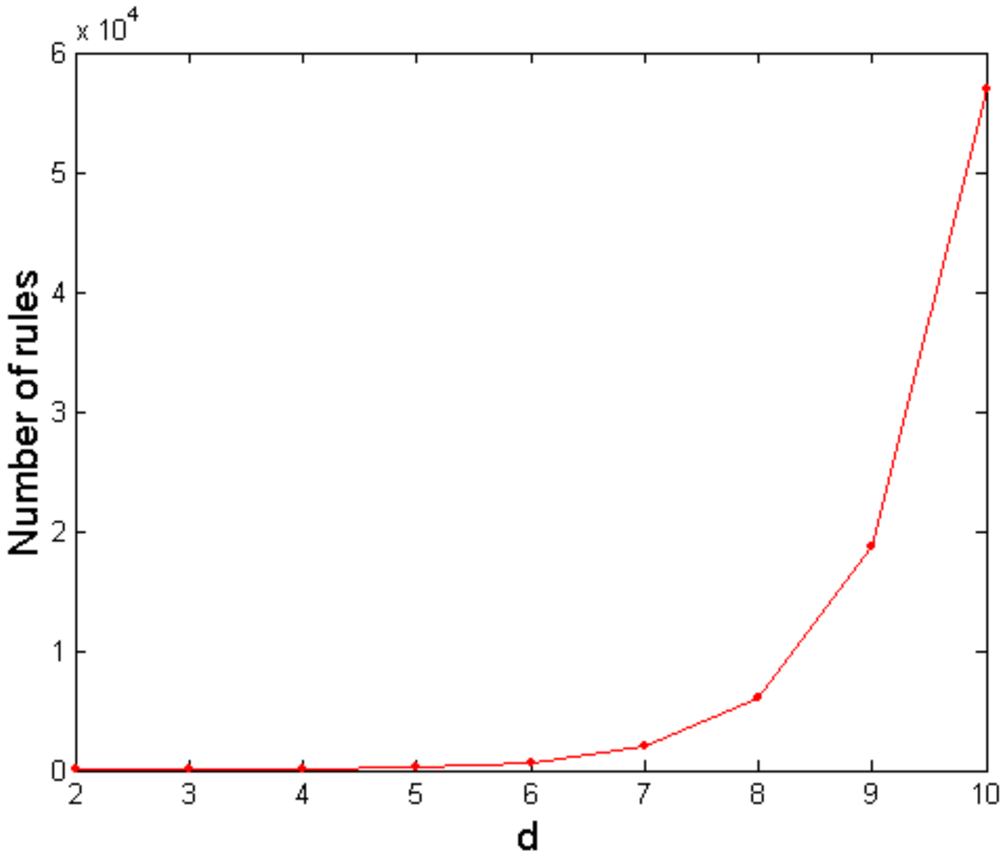
$$\text{Conf}(r) \geq M$$

Solução ingênua

- Enumerar todos os *itemsets* que aparecem nas transações de *D*:
  - Construir todos os sub-conjuntos de *D*.
  - Calcular o suporte de cada um destes subconjuntos.
  - Base de Dados = 1 milhão de transações.
  - Nº de items = 10000.
  - Nº de sub-conjuntos = 2^{10000} = ????
  - Nº de testes = $10^6 \times 2^{10000}$

Complexidade

- Dado *d* items:
  - número total de *itemsets* = 2^d
  - número total possível de regras de associação:



$$R = \sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right]$$
$$= 3^d - 2^{d+1} + 1$$

se d=6, R = 602 regras

se d=10, R= 57.002 regras

Algoritmo: duas etapas

1. Encontrar todos os *Itemset I* frequentes:

   $$\text{suporte}(I) \geq N$$

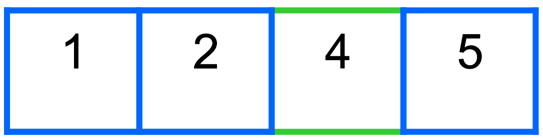
Etapa mais custosa – deve varrer a base de dados

2. $\text{Conf}(A, B \longrightarrow C) =$

   $$= \frac{\text{suporte}(A,B,C)}{\text{suporte}(A,B)} \geq M$$

Não há varredura da base de dados

Algoritmo *Apriori*



Propriedade Importante

Se um *itemset* é frequente

Todo *subitemset* é frequente !!

Algoritmo *Apriori* – etapa 1 encontrar *itemsets* frequentes

Entrada : BD de transações, N
Saida : F_1, F_2, F_3, …

C_1 = *Itemsets* de tamanho 1
F_1 = *Itemsets* frequentes de C_1
k : = 1
While F_k não for vazio
    C_{k+1} := Junta(F_k, F_k)
    C_{k+1} := Poda(C_k, F_k)
    F_{k+1} : = Valida(BD,C_{k+1}, N)
    k : = k+1

Estratégias para a geração de *itemsets* frequentes

- Reduzir o número de candidatos (*M*)
  - Busca completa: $M=2^d$.
  - Usar técnicas de poda para reduzir *M*.
- Reduzir o número de transações (*N*)
  - Reduzir o tamanho de *N* quando o número de itemsets aumenta.
  - Usado pelo DHP (*Direct Hasing and Prunning*) e algoritmos baseados em mineração vertical.
- Reduzir o número de comparações (*NM*)
  - Usar estruturas de dados eficientes para armazenar os candidatos ou as transações.
  - Sem necessidade de comparar cada candidato com cada transação.

Reduzindo o número de candidatos

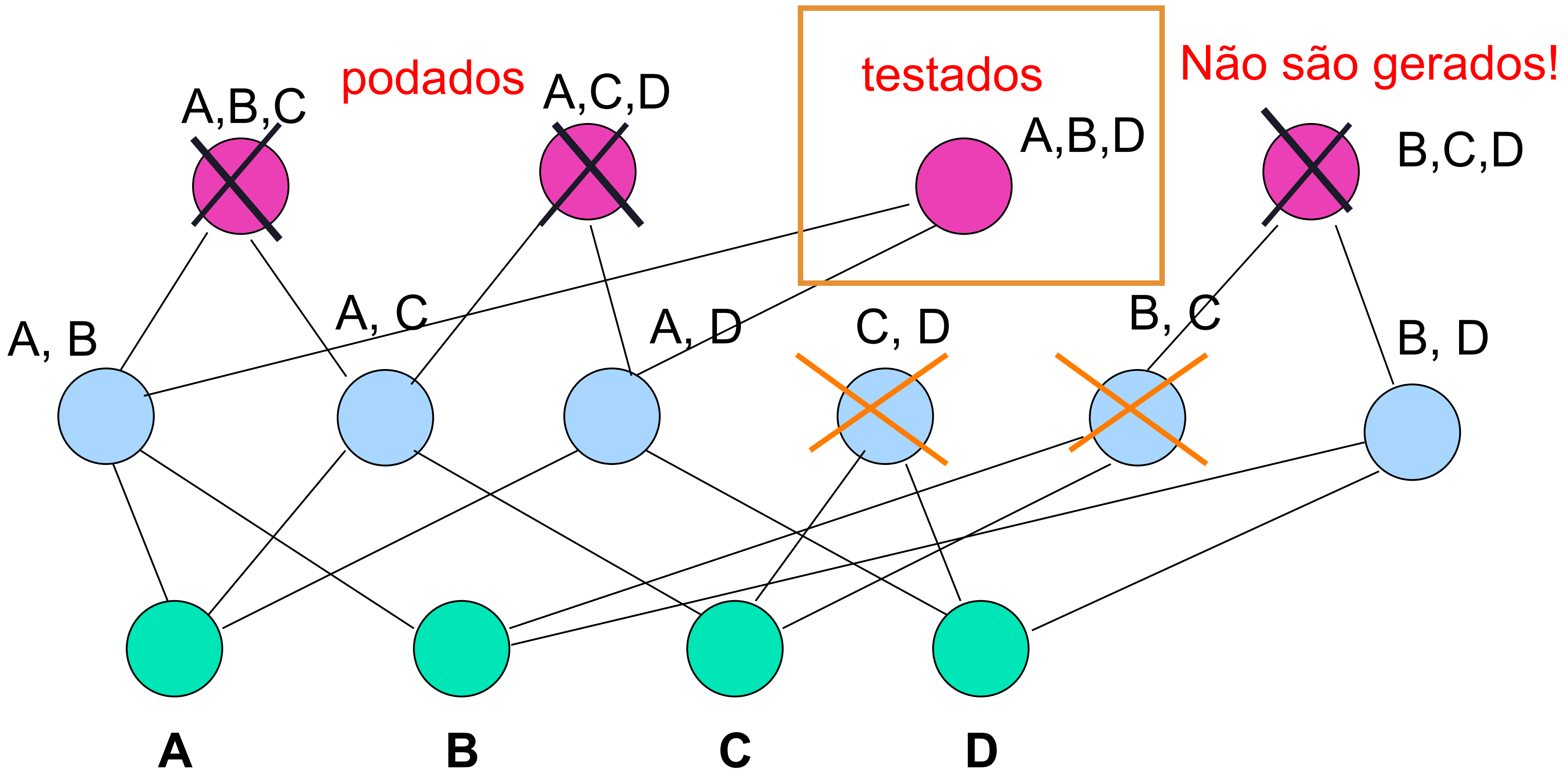
- Princípio do algoritmo *Apriori*:
  - Se um *itemset* é frequente então todos os seus subconjuntos também são frequentes.

- Este princípio é devido a seguinte propriedade do suporte:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - O suporte de um *itemset* nunca é maior que o suporte de seus subconjuntos.
  - Isto é conhecido como a propriedade anti-monotônica do suporte.

Ideia geral do algoritmo *Apriori*



Apriori – Fase da Geração

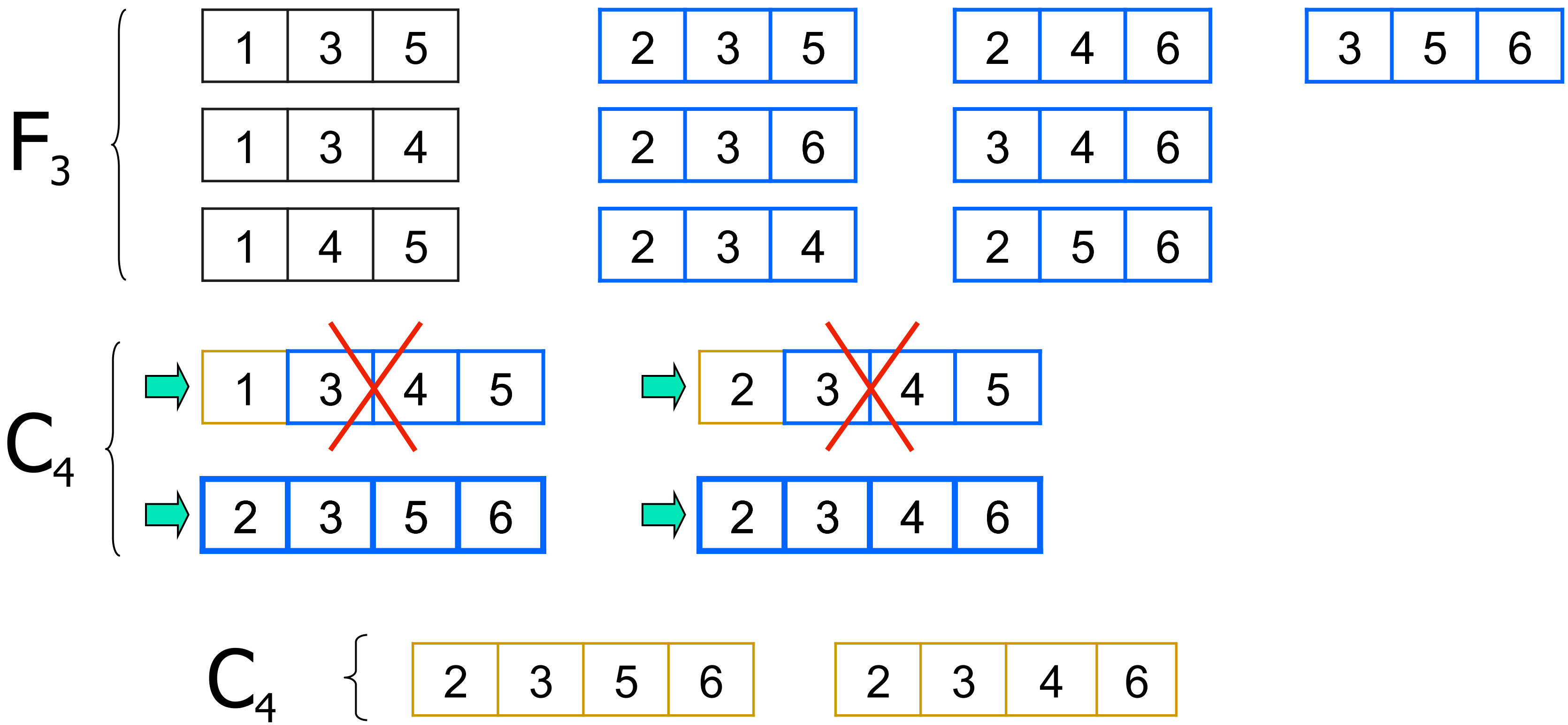
F_3

1 3 5	2 3 5	2 4 6	3 5 6
1 3 4	2 3 6	3 4 6	
1 4 5	2 3 4	2 5 6	

C_4

1 3 4 5	2 3 4 5
2 3 5 6	2 3 4 6

Apriori – Fase da Poda



Apriori – Fase de Validação

Suporte Mínimo: 50%

Candidatos	Contagem	Suporte
2 3 5 6	I	
2 3 4 6	III	

Banco de Dados
1 3 5 7 8
1 2 3 4 5 6 7
2 3 4 6 8
2 3 4 5 7 8
1 2 3 4 6 9

$F_4 = \{ \boxed{2\ 3\ 4\ 6} \}$

Um exemplo

Id	Compras
1	1,3,5
2	1,2,3,5,7
3	1,2,4,9
4	1,2,3,5,9
5	1,3,4,5,6,8
6	2,7,8

L1 = {1}, {2}, {3},{5}

C2 = {1,2} {1,3} {1,5} {2,3} {3,5} {2,5}

L2 = {1,2} {1,3} {1,5} {3,5}

C3 = {1,2,3} {1,2,5} {1,3,5}

L3 = {1,3,5}

Suporte minimo = 50%

Mineração de regras de associação

Dado um limite mínimo de confiança *M*

A regra X → Y é minerada se:

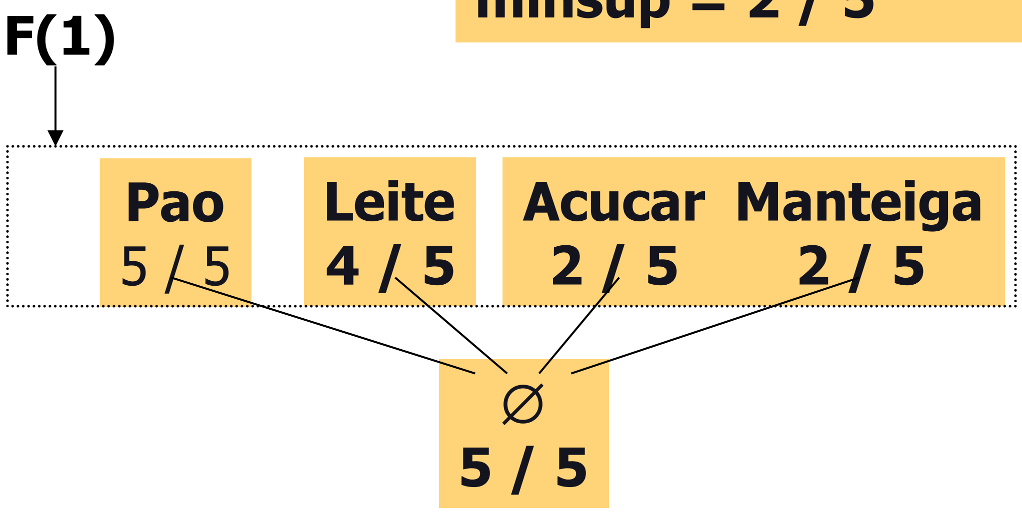
Suporte(X,Y) / Suporte(X) >= *M*

Exemplo completo

- Cálculo de F(1)

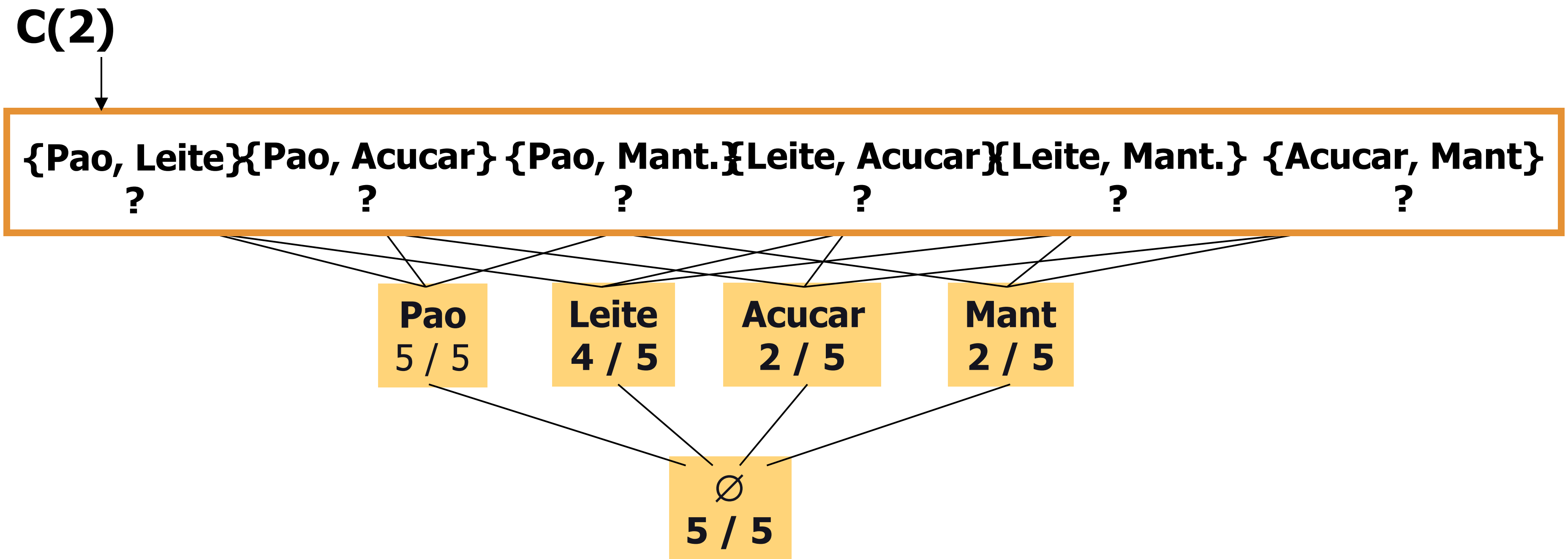
T1 = { Pao, Leite, Manteiga }
T2 = { Pao, Leite, Acucar }
T3 = { Pao}
T4 = { Pao, Leite }
T5 = { Pao, Leite, Manteiga, Acucar }

minsup = 2 / 5



Geração de C(2)

- Combinação dos elementos de F(1)
- Poda dos elementos de C(2) – nenhuma neste nível

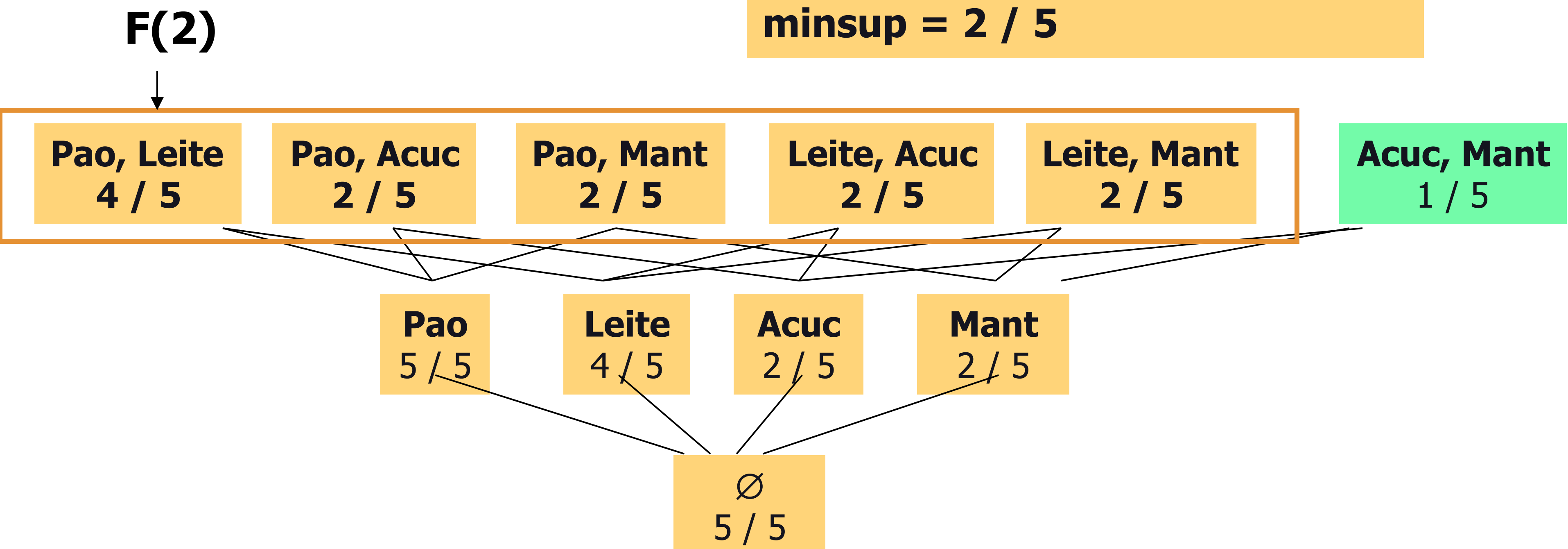


Cálculo do suporte de C(2)

- Cálculo de F(2)

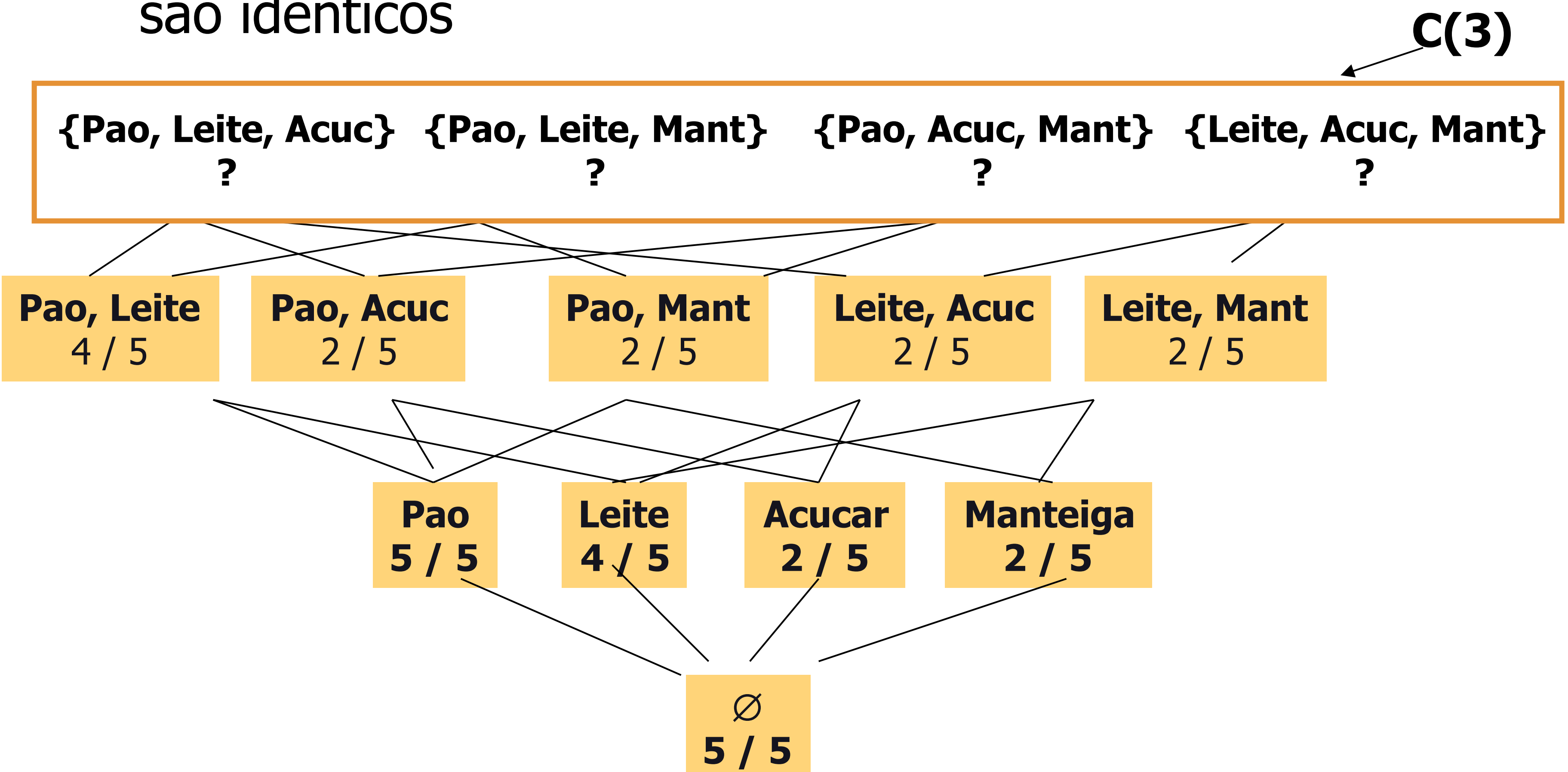
Numa varrida dos dados

T1 = { Pao, Leite, Mant }
T2 = { Pao, Leite, Acuc }
T3 = { Pao }
T4 = { Pao, Leite }
T5 = { Pao, Leite, Mant, Acuc }
minsup = 2 / 5

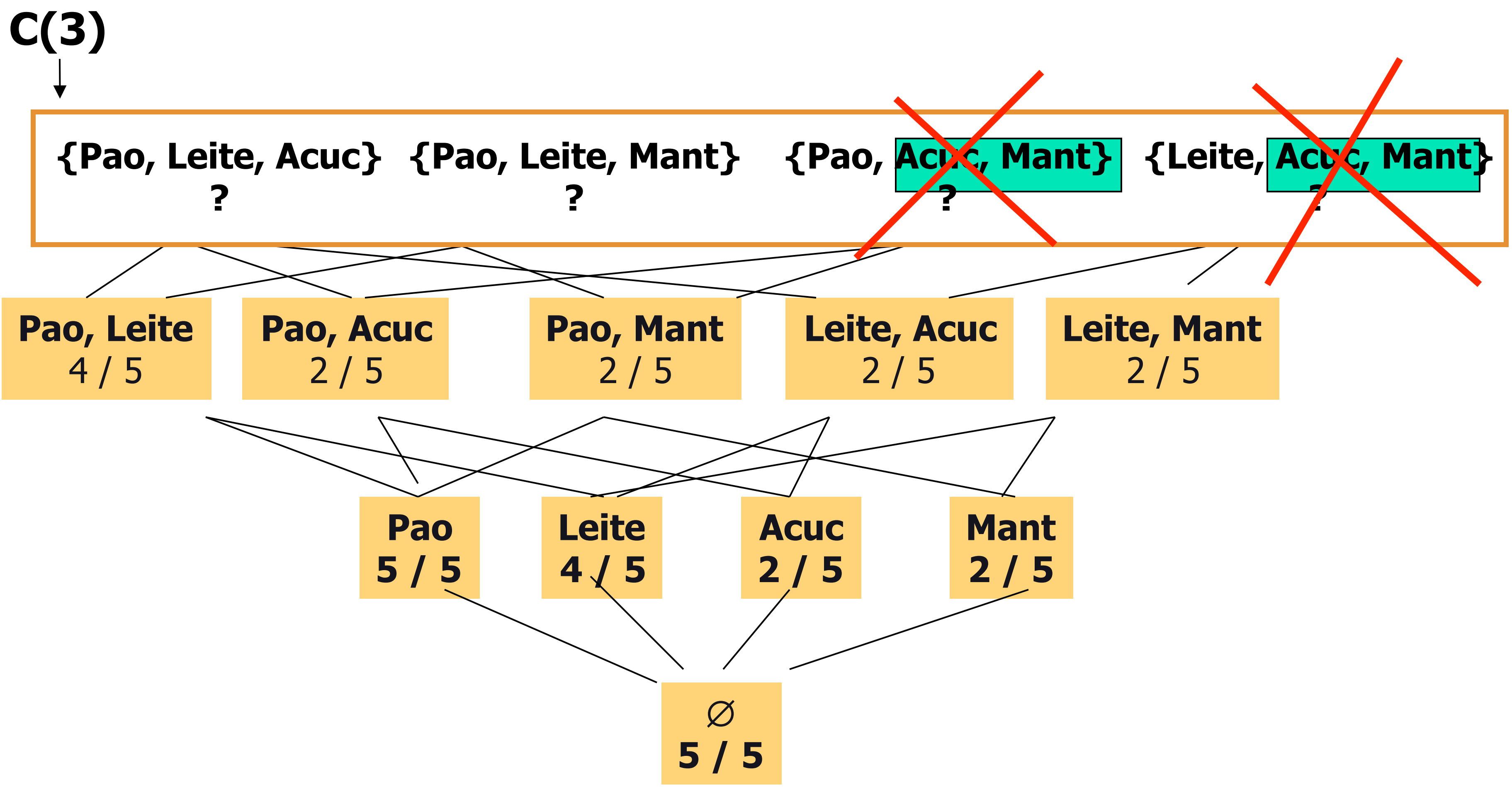


Geração de C(3)

Combinar somente os itemsets cujos primeiros elementos sao idênticos

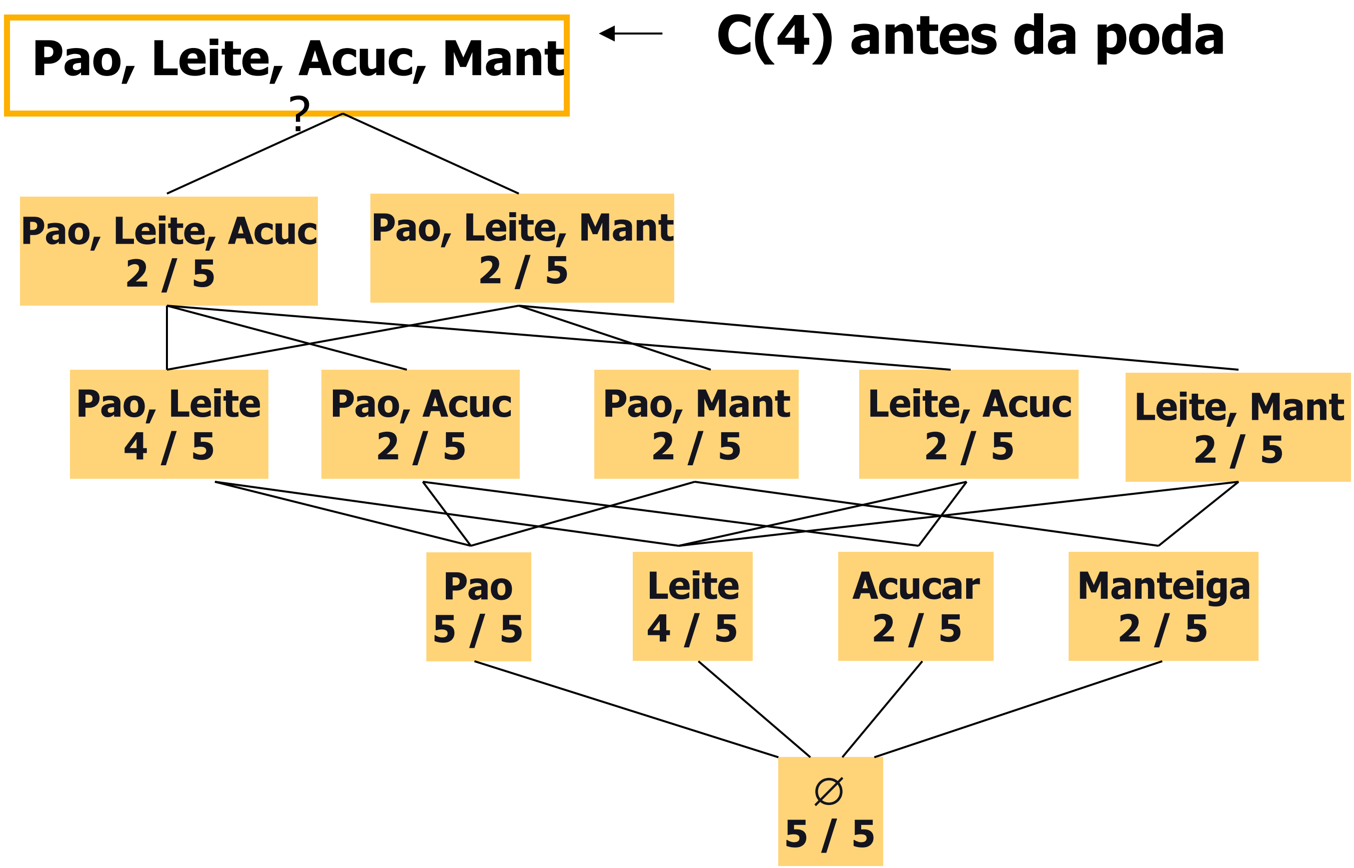


Poda dos elementos de C(3)

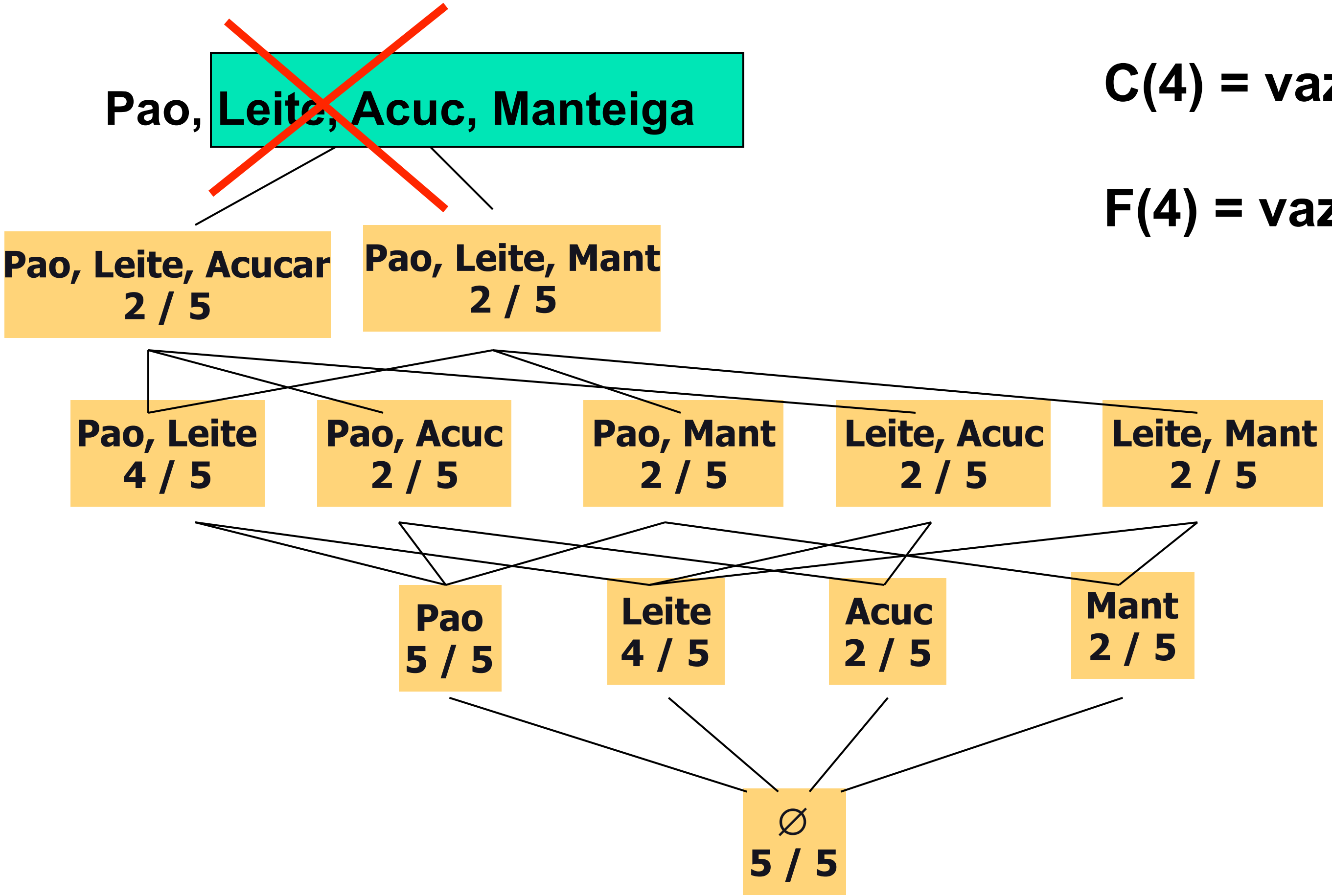


Geração de C(4)

- Combinação dos elementos de F(3)



Poda dos elementos de C(4)



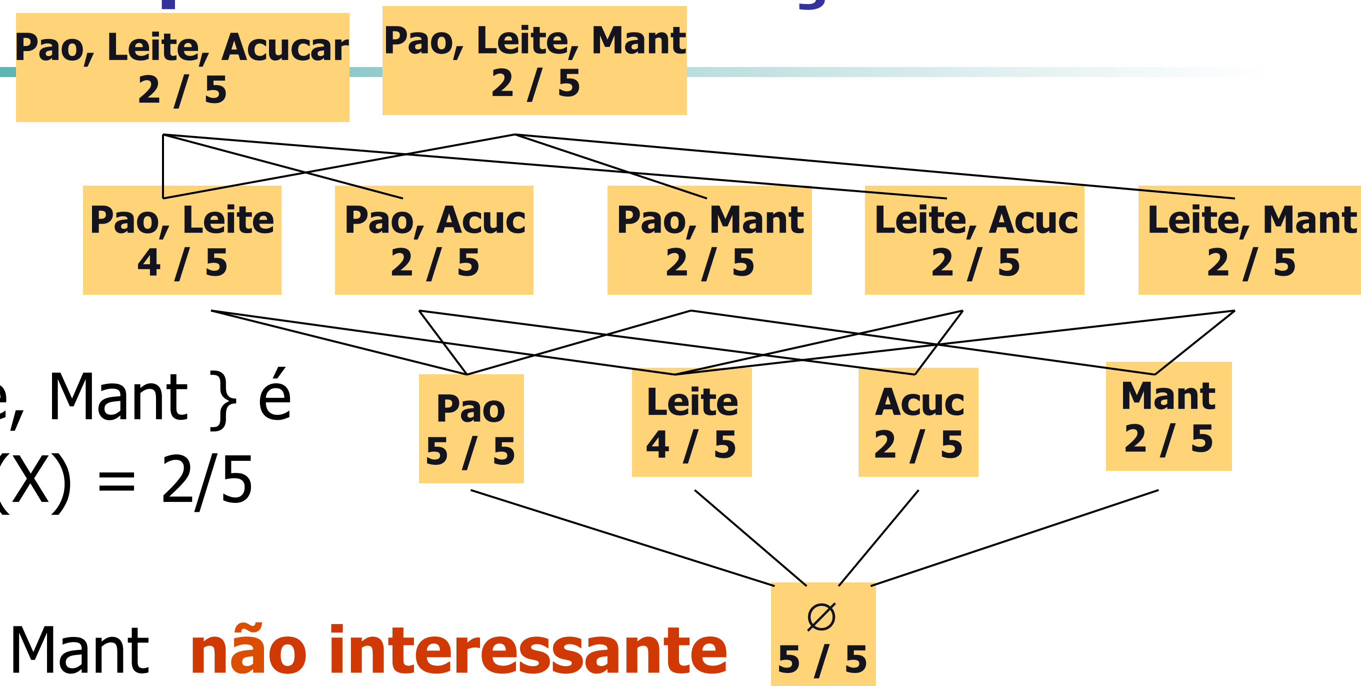
C(4) = vazio

F(4) = vazio

Cálculo das regras interessantes

- Para todo *k-itemset* X frequente
  (com $k > 1$) e $Y \subset X$
  - Calcular a *confiança* da regra de associação
    $$X - Y \Rightarrow Y$$
    - Se é superior ou igual a *minconf*, então a regra gerada é **interessante**
  - **Observação**
    - **sup ((X-Y) U Y) = sup(X) > minsup**
    - **Conf(X-Y ⇒ Y) =** $\dfrac{\textbf{support(X)}}{\textbf{support (X-Y)}}$

Exemplo - continuação



minconf = 3 / 4

X = { Pao, Leite, Mant } é frequente   sup(X) = 2/5

- Pao, Leite ⇒ Mant **não interessante**
  sup (Pao, Leite) = 4/5, Conf = (2/5) / (4/5) = 2/4
- Pao, Mant ⇒ Leite **interessante, pois** Conf = 2 / 2
- Leite, Mant ⇒ Pao **interessante, pois** Conf = 2 / 2
- Pao ⇒ Leite, Mant **não interessante, pois** Conf = 2 / 5
- Leite ⇒ Pao , Mant **não interessante, pois** Conf = 2 / 4
- Mant ⇒ Pao , Leite **não interessante, pois** Conf = 2 / 4

Problema: número de regras geradas

Considerando 4 itens: A, B, C e D, sem considerar suporte e confiança, podemos ter:

Sets	*Possible Rules*	*Number of Rules*
{AB}	*A→B; B→A*	*2*
{AC}	*A→C; C→A*	*2*
{AD}	*A→D; D→A*	*2*
{BC}	*B→C; C→B*	*2*
{BD}	*B→D; D→B*	*2*
{CD}	*C→D; D→C*	*2*
{ABC}	*A→BC; B→AC; C→AB; BC→A; AC→B; AB→C*	*6*
{ABD}	*A→BD; B→AD; D→AB; BD→A; AD→B; AB→D*	*6*
{ACD}	*A→DC; D→AC; C→AD; DC→A; AC→D; AD→C*	*6*
{BCD}	*D→BC; B→DC; C→DB; BC→D; DC→B; DB→C*	*6*
{ABCD}	*A→BCD; B→ACD; C→ABD; D→ABC; AB→CD; AC→BD; AD→BC; BC→AD; BD→AC; CD→AB; BCD→A; ACD→B; ABD→C; ABC→D;*	*14*

Reduzindo o número de regras

- ***Suporte*** e ***confiança*** são usados como filtros, para diminuir o número de regras geradas, gerando apenas regras de melhor qualidade.

- Mas, se considerarmos a regra:

  `Se A então B` com confiança de 90%

- Podemos garantir que seja uma regra interessante?

LIFT

- A regra (1) **`Se A então B`** com confiança de 90% **NÃO** é interessante se B aparece em cerca de 90% das transações, pois a regra não acrescentou nada em termos de conhecimento.

- Já a regra (2): **`Se C então D`** com confiança de 70% é muito mais importante se D aparece, digamos, em 10% das transações.

  lift = confiança da regra / suporte do consequente

  lift da regra (1) = 0,9 / 0,9 = 1

  lift da regra (2) = 0,7 / 0,1 = 7

LIFT

Transaction 1	🍎 🍺 🍚 🍗
Transaction 2	🍎 🍺 🍚
Transaction 3	🍎 🍺
Transaction 4	🍎 🍐
Transaction 5	🍼 🍺 🍚 🍗
Transaction 6	🍼 🍺 🍚
Transaction 7	🍼 🍺
Transaction 8	🍼 🍐

$$\text{Lift } \{🍎 \rightarrow 🍺\} = \frac{\text{Support } \{🍎, 🍺\}}{\text{Support } \{🍎\} \times \text{Support } \{🍺\}}$$

Regras redundantes

Tid	Itemset
1	A, C, D,T, W
2	C, D, W
3	A, D, T, W
4	A, C, D, W
5	A, C, D, T, W
6	C, D, T

A→W s=4/6 c=4/4

A→D,W s=4/6 c=4/4

Organizando os conjuntos frequentes por transações

TidSet	Frequent sets *L*
123456	**{D}**
12456	{C}, **{C,D}**
12345	{W}, **{D,W}**
1245	{C,W}, **{C,D,W}**
1345	{A}, {A,D}, {A,W}, **{A,D,W}**
1356	{T}, **{D,T}**
145	{A,C}, {A,C,W}, {A,C,D}, **{A,C,D,W}**
135	{A,T}, {T,W}, {A,D,T}, {A,T,W}, {D,T,W}, **{A,D,T,W}**
156	{C,T}, **{C,D,T}**

Conjuntos fechados (*Closed Itemsets*)

- Um conjunto de itens (*itemset*) é fechado se nenhum de seus superconjuntos imediatos tem o mesmo suporte que ele (nas mesmas transações)

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

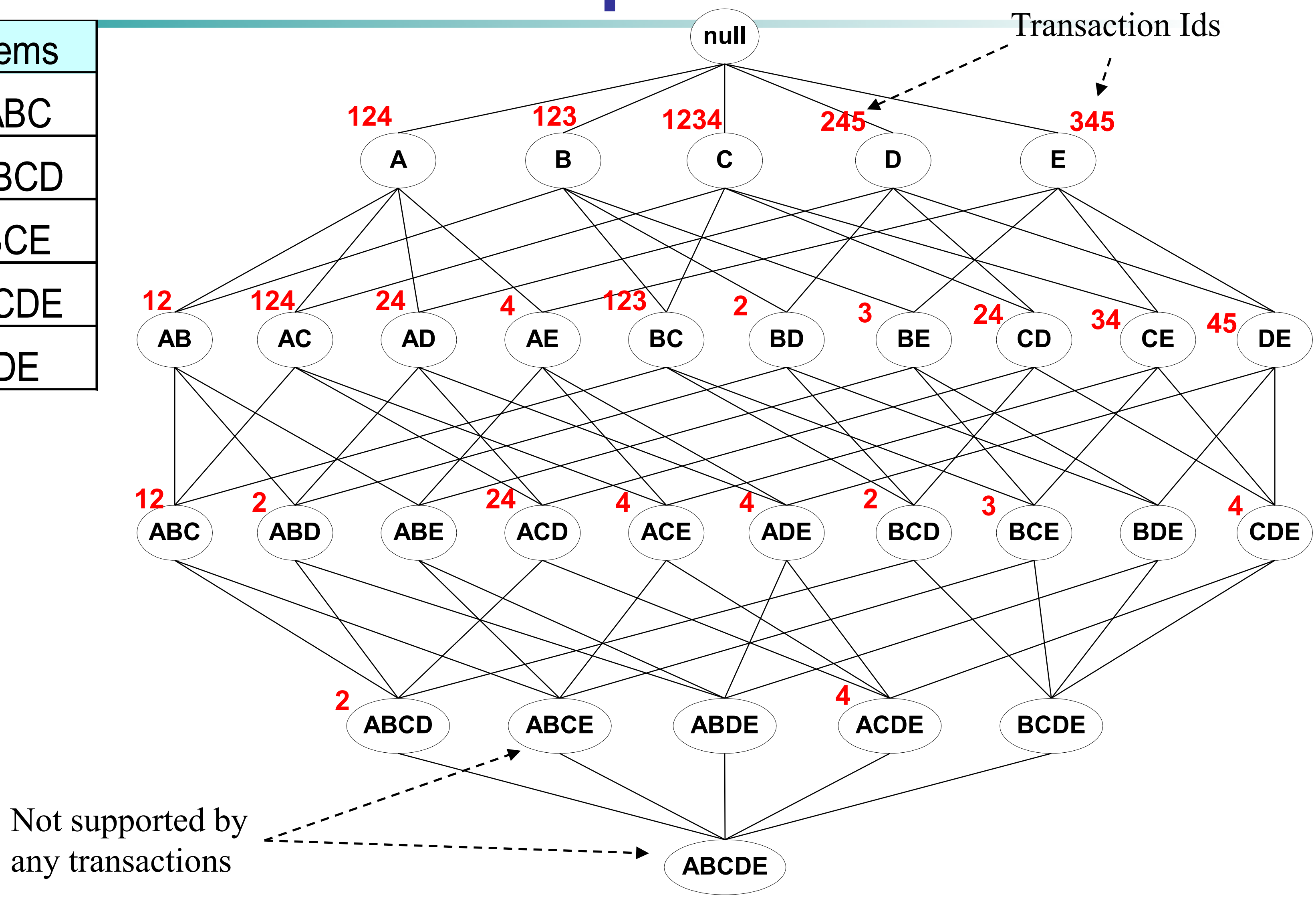
Para suporte minimo 3, conjuntos
Marcados são fechados

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

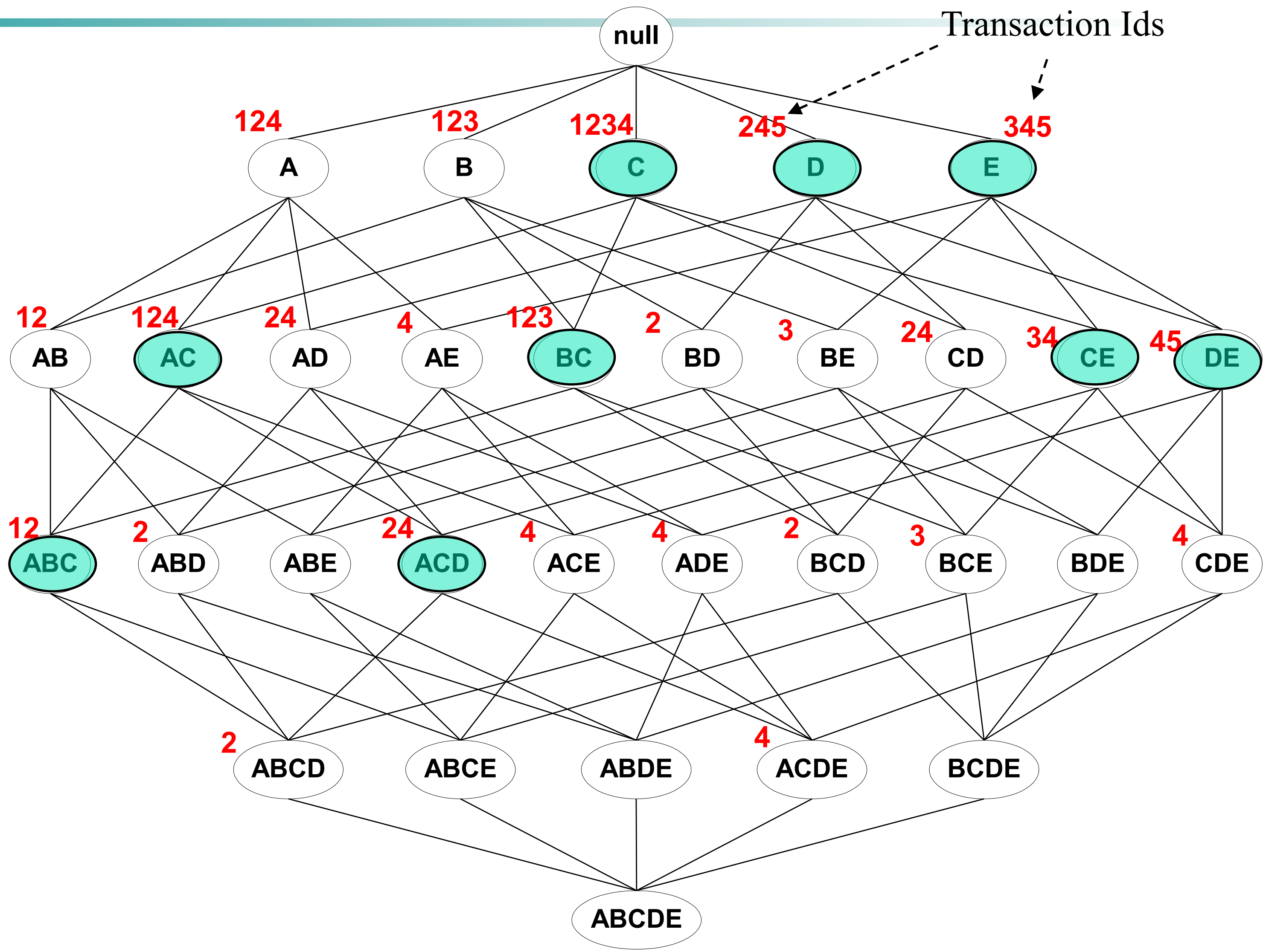
14 Conjuntos frequentes com minsup=2

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



9 Conjuntos fechados (para minsup=2)

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Regras de Associação (RA)

A→W S=4/6 C=100%

A→DW

Conjuntos frequentes geram RA redundantes

Set *k*	Conjuntos frequentes *minsup* 50%
k=1	{A}, {C}, {D}, {T}, {W}
k=2	{A,C}, {A,D}, {A,T}, {A,W}, {C,D}, {C,T}, {C,W}, {D,T}, {D,W}, {T,W}
k=3	{A,C,D}, {A,C,W}, {A,D,T}, {A,D,W}, {A,T,W}, {C,D,T}, {C,D,W}, {D,T,W}
k=4	{A,C,D,W}, {A,D,T,W}

Tid	Itemset
1	A, C, D, T, W
2	C, D, W
3	A, D, T, W
4	A, C, D, W
5	A, C, D, T, W
6	C, D, T

A→DW S=4/6 C=100%

Set *k*	Conjuntos frequentes fechados *minsup* 50%
k=1	{D}
k=2	{C,D}, {D,T}, {D,W},
k=3	{A,D,W}, {C,D,T}, {C,D,W}
k=4	{A,C,D,W}, {A,D,T,W}

Conjuntos frequentes fechados reduzem RA redundantes

Gargalos de performance no *Apriori*

- O núcleo do algoritmo:
  - Usa (k – 1)-*itemsets* frequentes para gerar *k-itemsets* candidatos.
  - Usa iterações pelo BD e casamento de padrões para coletar contadores para os *itemsets* candidatos.
  - O gargalo do *Apriori*: geração dos candidatos.
  - Grandes conjuntos de candidatos:
    - 10^4 1-*itemset* frequentes gerarão 10^7 2-*itemsets* candidatos.
    - Para descobrir um padrão frequente de tamanho 100, é necessária a geração de $2^{100} \approx 10^{30}$ candidatos.
  - Múltiplas iterações pelo BD:
    - Necessita ($n + 1$) iterações, onde n é o tamanho do maior padrão.

Métodos para melhorar a eficiência do *Apriori*

- Contagem dos itemsets baseada em Hashes: Um *k-itemset* que tenha o contador do *hashing bucket* abaixo de um limite, não pode ser frequente.
- Redução de transações: Uma transação que não contenha nenhum *k-itemset* frequente, é inútil para as próximas iterações do algoritmo.
- Particionamento: Qualquer *itemset* que é potencialmente frequente no BD deve ser frequente em pelo menos uma partição do mesmo.
- Amostragem: Mineração em um subconjunto dos dados, menor limite de suporte + um método para determinar a completude.
- Contagem dinâmica de itemsets: Adicionar um novo candidato somente quando todos os seus subconjuntos são estimados como frequentes.

Algoritmo *FP-Growth*

- O algoritmo *Apriori* pode sofrer dois problemas:
  - dificuldade para tratar uma grande quantidade de conjuntos candidatos;
  - execução de repetidas passagens pela base de dados.

- Para mitigar tais problemas o algoritmo *FP-Growth* (*Frequent Pattern Growth*) é baseado em uma estrutura em árvore de prefixos para os padrões frequentes, denominada *FP-Tree* (*Frequent Pattern Tree*), a qual armazena de forma comprimida a informação sobre os padrões frequentes.
- O algoritmo *FP-Growth* extrai o conjunto completo de padrões frequentes.

Algoritmo *FP-Growth*

- A essência do algoritmo proposto está baseada em três aspectos centrais:
  - A compressão da base de dados em uma estrutura em árvore (*FP-Tree*) cujos nós possuem apenas itens frequentes de comprimento unitário (F_1) e organizada de modo que aqueles nós que ocorrem mais frequentemente terão maiores chances de compartilhar nós do que os de baixa frequência.
  - O uso de um algoritmo de mineração da árvore que evita a geração de uma grande quantidade de conjuntos candidatos.
  - O uso de um método particional para decompor a tarefa de mineração em subtarefas menores, reduzindo significativamente o espaço de busca.

Algoritmo *FP-Growth*

- *FP-Growth*: permite a descoberta de *itemsets* frequentes sem a geração de conjunto de itens candidatos. Abordagem em duas etapas:
  - Passo 1: Construa uma estrutura de dados compacta chamada *FP-Tree*.
    - Construída usando duas passagens pelo conjunto de dados.
  - Passo 2: Extrai *itemsets* frequentes diretamente da *FP-Tree*.

Passo 1: Construção da *FP-Tree*

- *FP-Tree* é construída usando dois passos sobre o conjunto de dados:

  Passo 1:
  - Varrer os dados e encontrar o suporte para cada um deles.
  - Descartar itens não frequentes.
  - Ordenar items frequentes em ordem descendente baseado em seu suporte.

Usar esta ordem quando construer a *FP-Tree*, assim prefixos comuns podem ser compartilhados.

Passo 1: Construção da *FP-Tree*

Passo 2:

Nós correspondem a items e têm um contador:

1. *FP-Growth* lê uma transação de cada vez e mapeia-a para um caminho.
2. A ordem fixada é usada, assim caminhos podem se sobrepor quando transações compartilham items (quando eles têm o mesmo prefixo).
   - Neste caso, contadores são incrementados.
3. Ponteiros são mantidos entre nós contendo o mesmo item, criando listas ligadas isoladamente (linhas pontilhadas).
   - Quanto mais os caminhos se sobrepõem, mais alta a compressão. *FP-Tree* pode caber na memória.
4. *Itemsets* frequentes extraídos da *FP-Tree*.

Exemplo de uma *FP-Tree*

Transações

A B C E F O
A C G
E I
A C D E G
A C E G L
E J
A B C E F P
A C D
A C E G M
A C E G N

Freq. 1-Itemsets.
Supp. Count ≥2

A:8	
C:8	
E:8	
G:5	
B:2	
D:2	
F:2	

Transações com itens baseados em frequências,ignorando os itens não frequentes

A C E B F
A C G
E
A C E G D
A C E G
E
A C E B F
A C D
A C E G
A C E G

FP-Tree após leitura da 1ª transação

A C E B F

A C G

E

A C E G D

A C E G

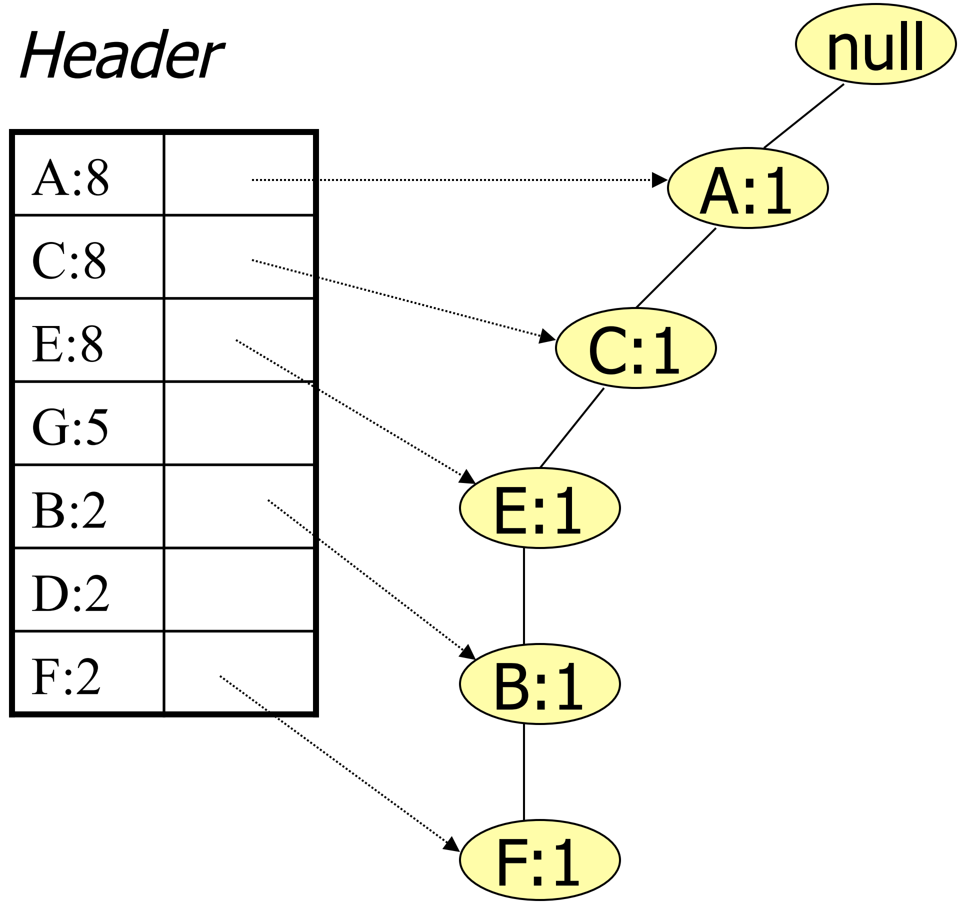
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 2ª transação

A C E B F

A C G

E

A C E G D

A C E G

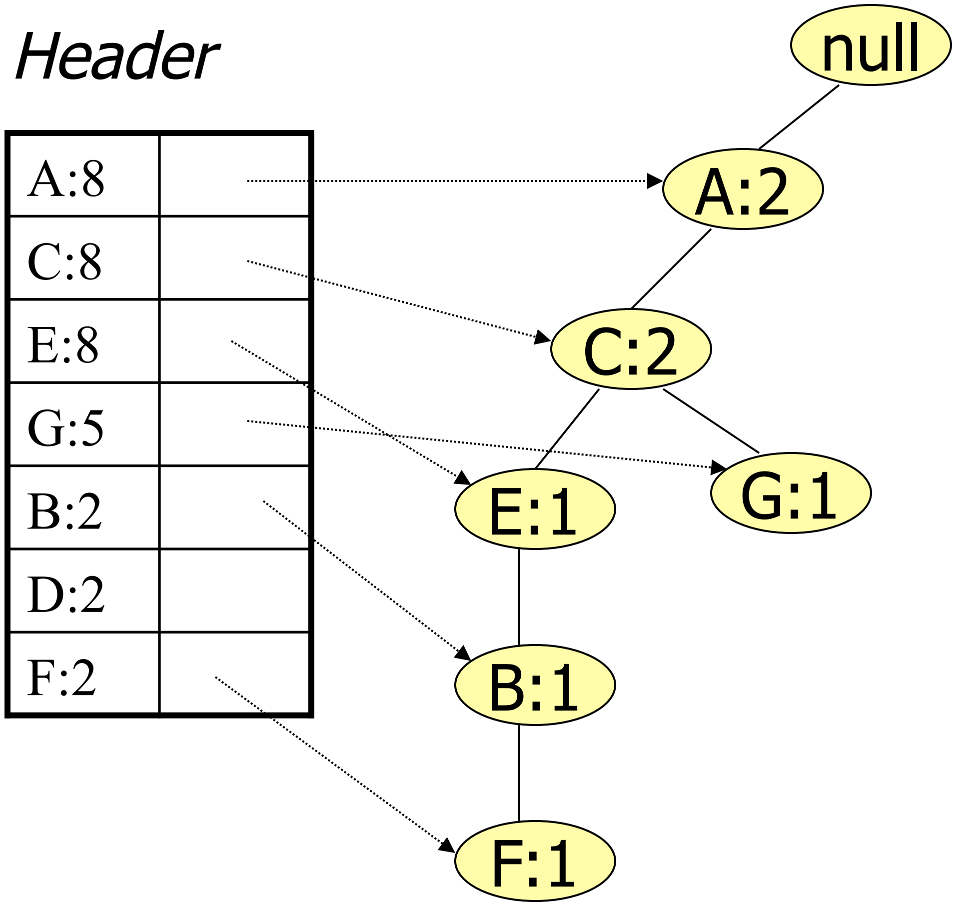
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 3ª transação

A C E B F

A C G

E

A C E G D

A C E G

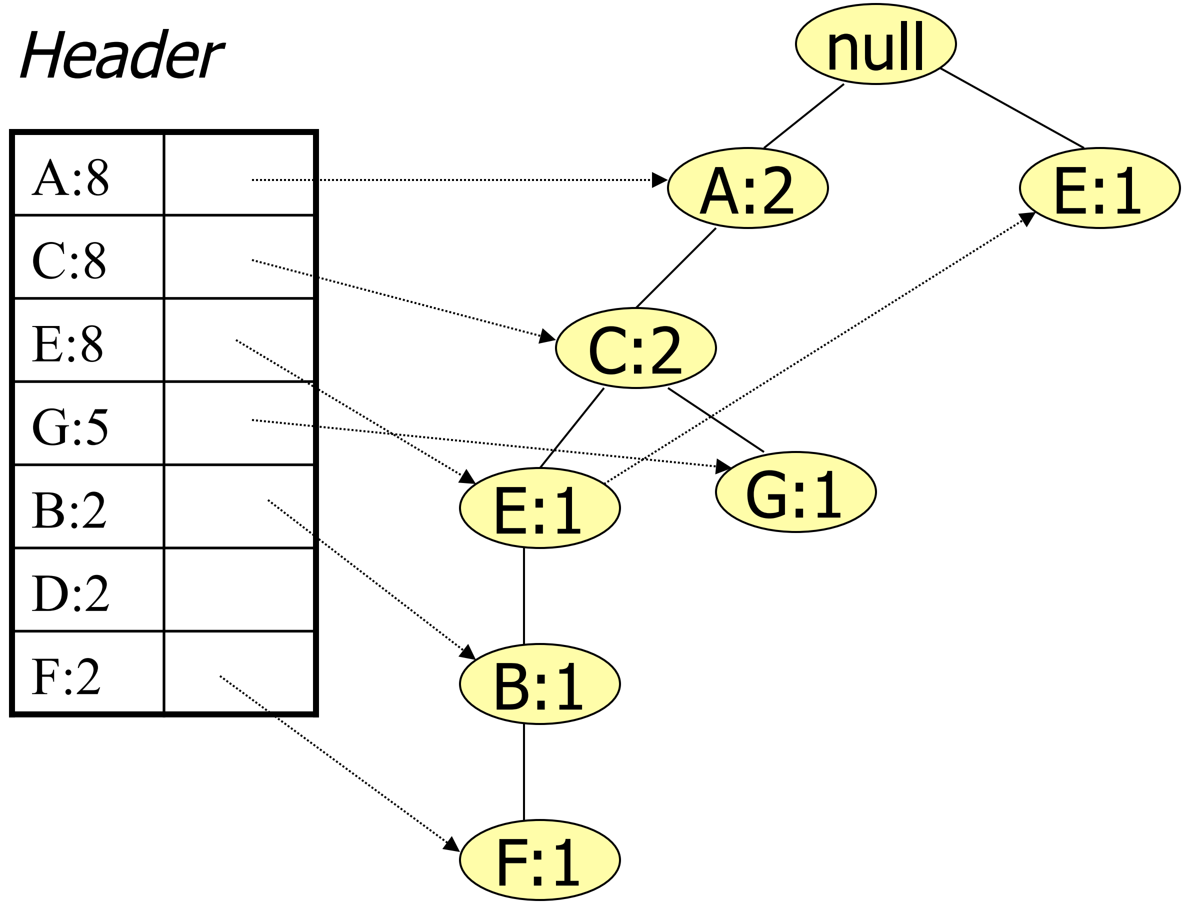
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 4ª transação

A C E B F

A C G

E

A C E G D

A C E G

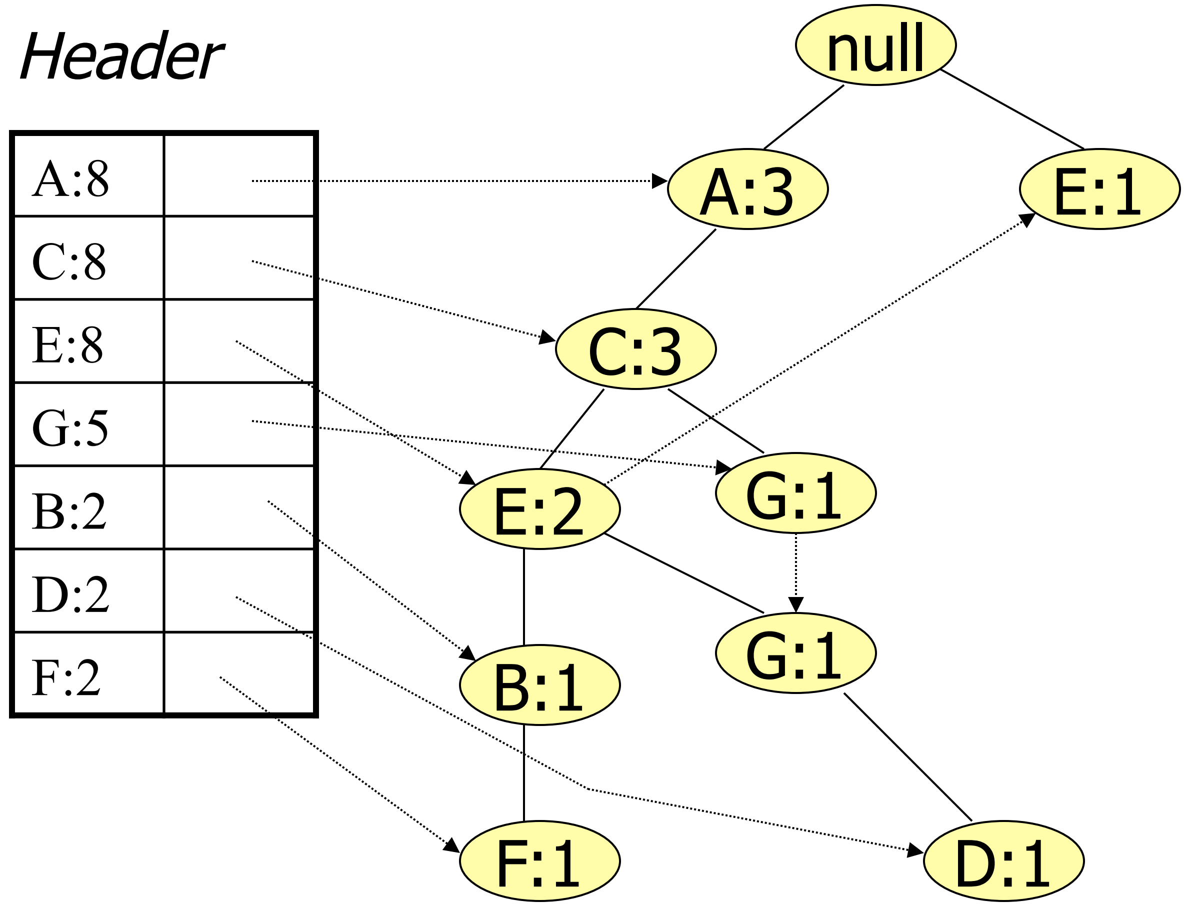
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 5ª transação

A C E B F

A C G

E

A C E G D

A C E G

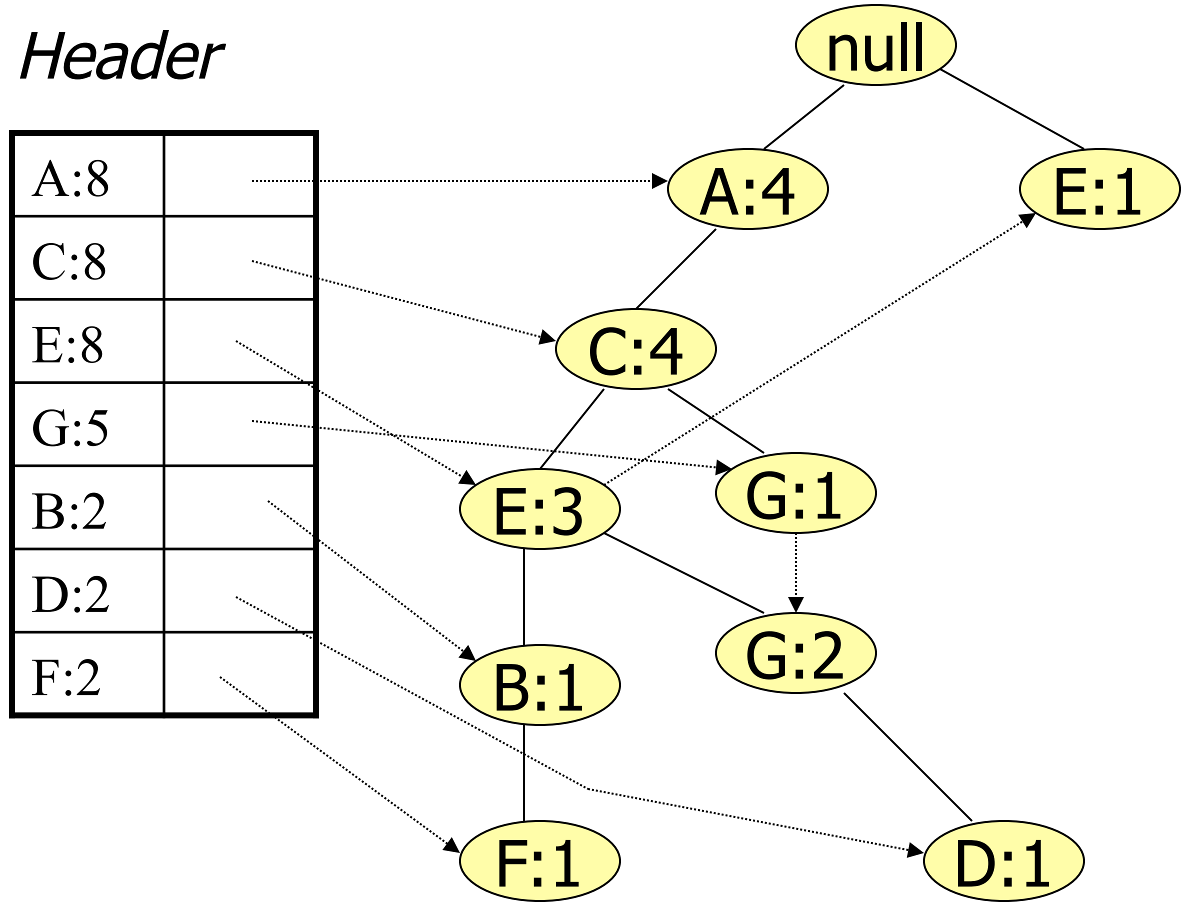
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 6ª transação

A C E B F

A C G

E

A C E G D

A C E G

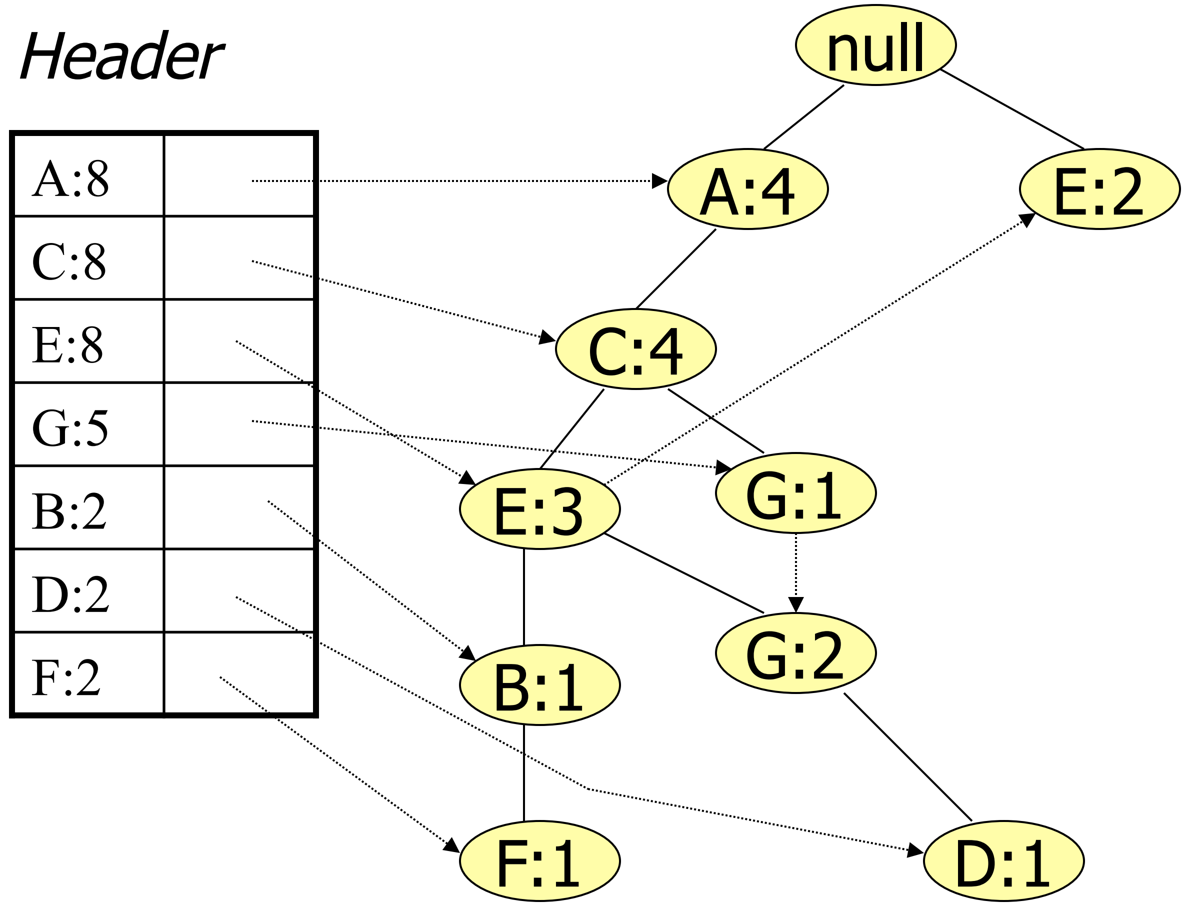
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 7ª transação

A C E B F

A C G

E

A C E G D

A C E G

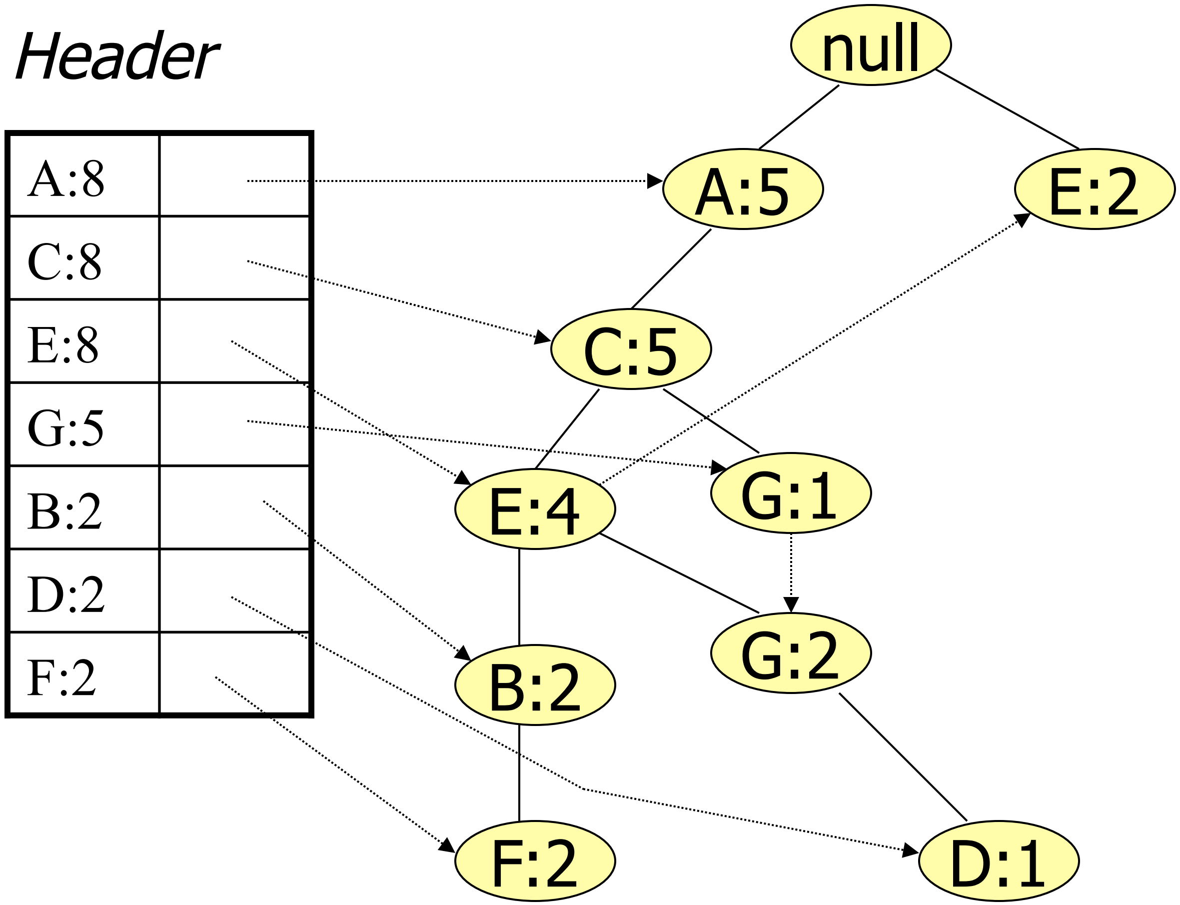
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 8ª transação

A C E B F

A C G

E

A C E G D

A C E G

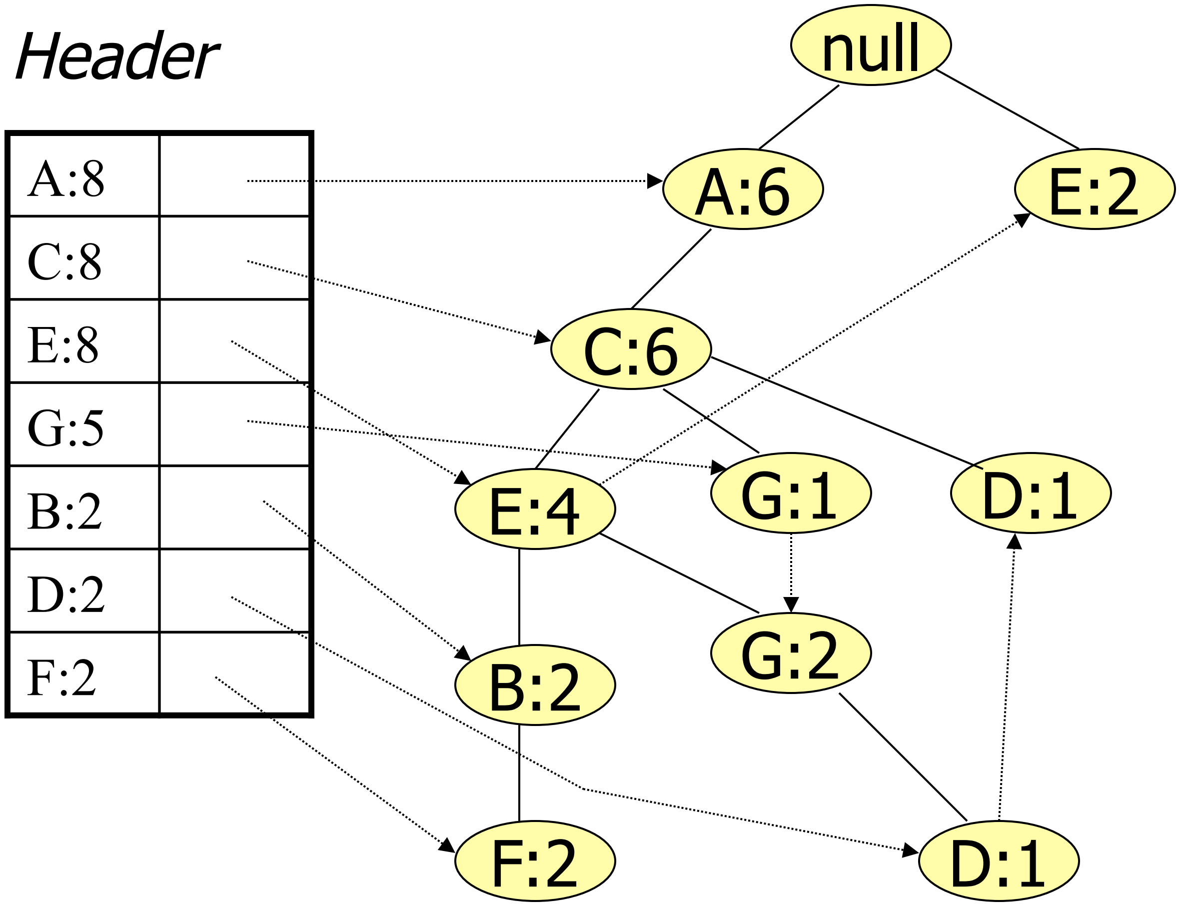
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 9ª transação

A C E B F

A C G

E

A C E G D

A C E G

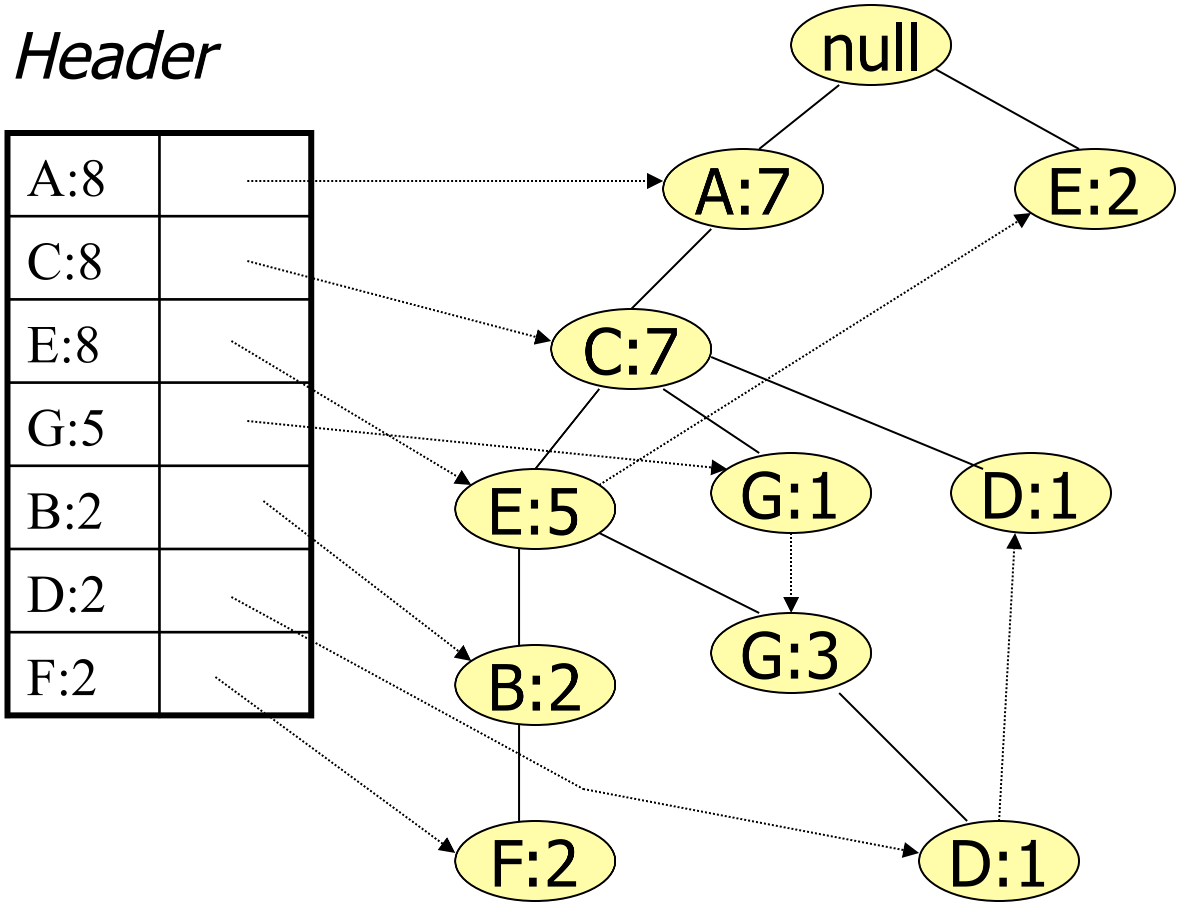
E

A C E B F

A C D

A C E G

A C E G



FP-Tree após leitura da 10ª transação

A C E B F

A C G

E

A C E G D

A C E G

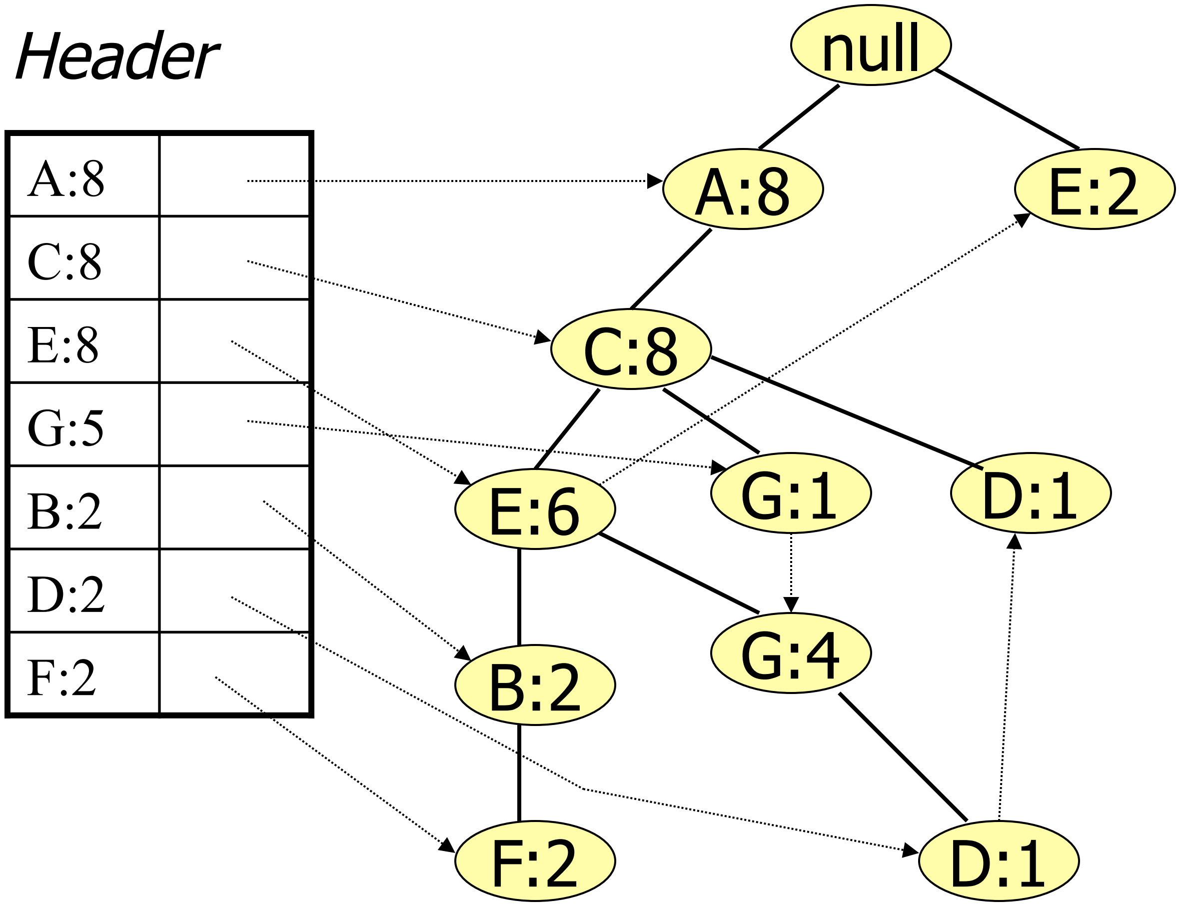
E

A C E B F

A C D

A C E G

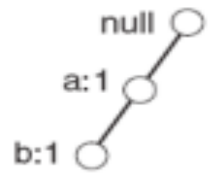
A C E G



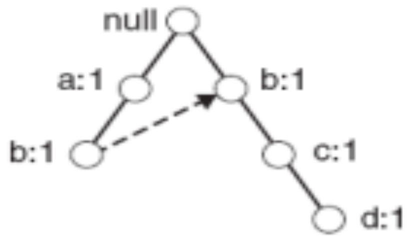
Passo 1: Construção da *FP-Tree* (Exemplo)

Transaction Data Set

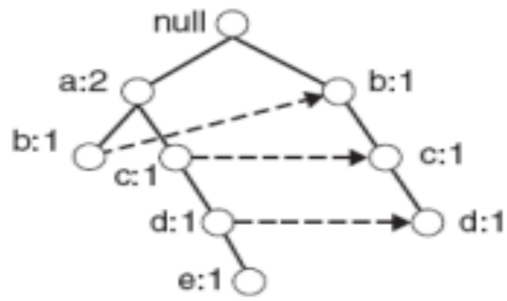
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



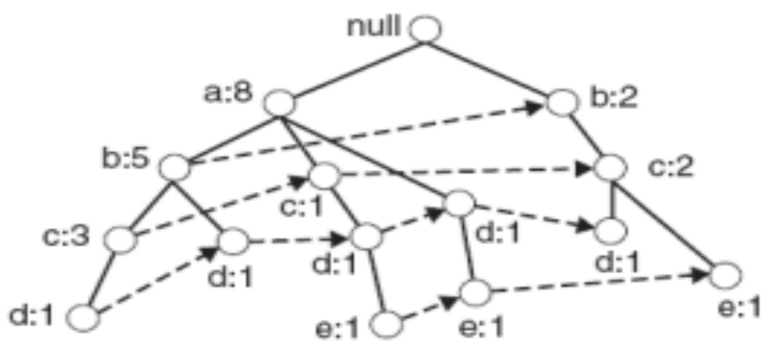
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3



(iv) After reading TID=10

Tamanho da *FP-Tree*

- Uma *FP-Tree* ugeralmente tem um tamanho menor que os dados não comprimidos – tipicamente muitas transações compartilham items (e até prefixos).
  - Cenário do melhor caso: todas as transações contêm o mesmo conjunto de itens.
    - Um caminho na *FP-Tree*.
  - Cenário do pior caso: cada transação tem um único conjunto de itens (sem itens em comum).
    - O tamanho da *FP-Tree* é pelo menos tão larga quanto os dados originais.
    - Requisitos de armazenamento para a *FP-Tree* são mais altos – precisam armazenar os ponteiros entre os nós e os contadores.
- O tamanho da *FP-Tree* depende de como os itens estão ordenados.
- Ordernação por suporte descendente é tipicamente usado mas nem sempre leva à menor árvore (é uma heurística).

Passo 2: Geração do *Itemset* frequente

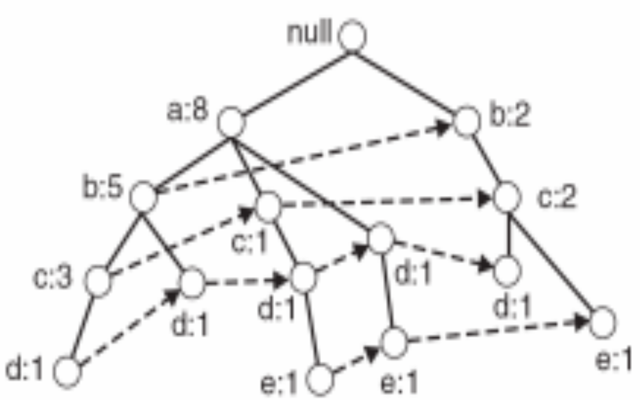
- O *FP-Growth* extrai *itemsets* frequentes da *FP-Tree*.

- Algoritmo *bottom-up* - das folhas em direção à raiz.

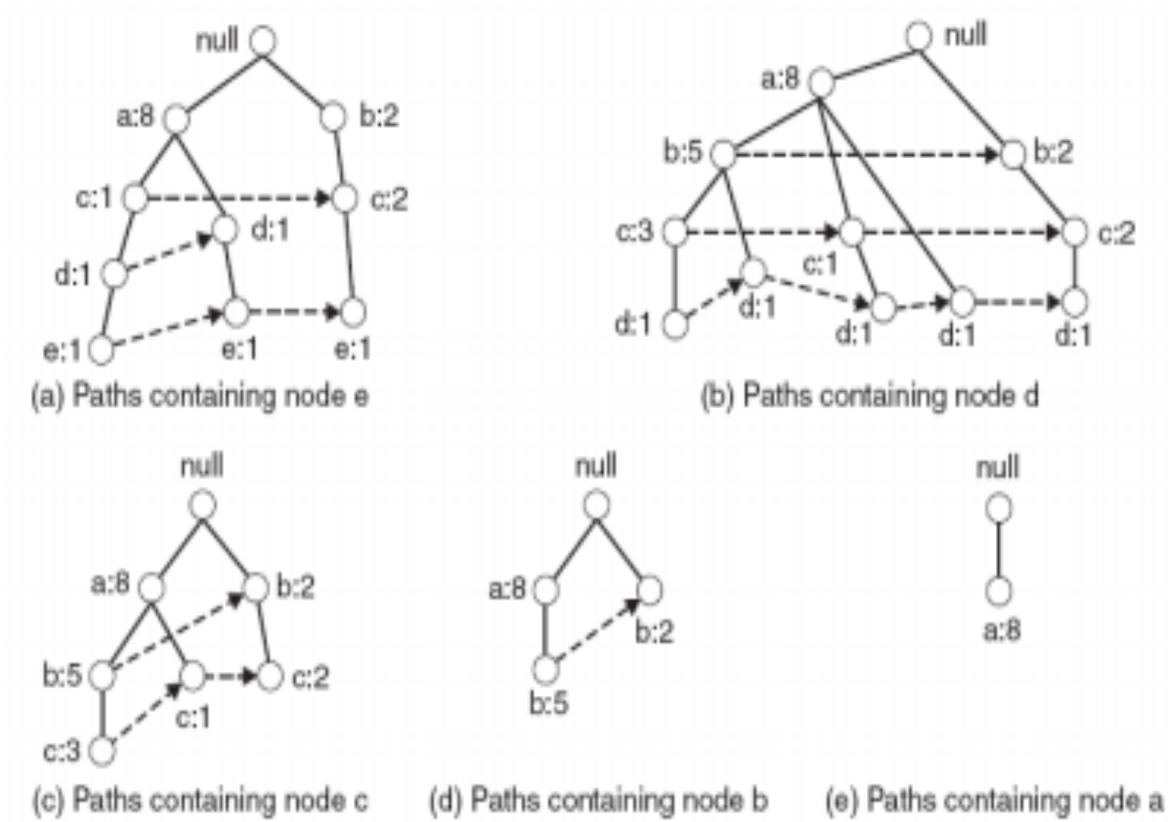
- Dividir e conquistar: primeiro procure por *itemsets* frequentes que terminem em *e*, depois *de*, etc. então *d*, depois *cd*, etc.

- Primeiro, extraia as subárvores do caminho de prefixo que terminam em um item (conjunto). (sugestão: use as listas ligadas).

Sub-árvores do cominho de prefixo (Exemplo)



↑ Complete FP-tree



(a) Paths containing node e

(b) Paths containing node d

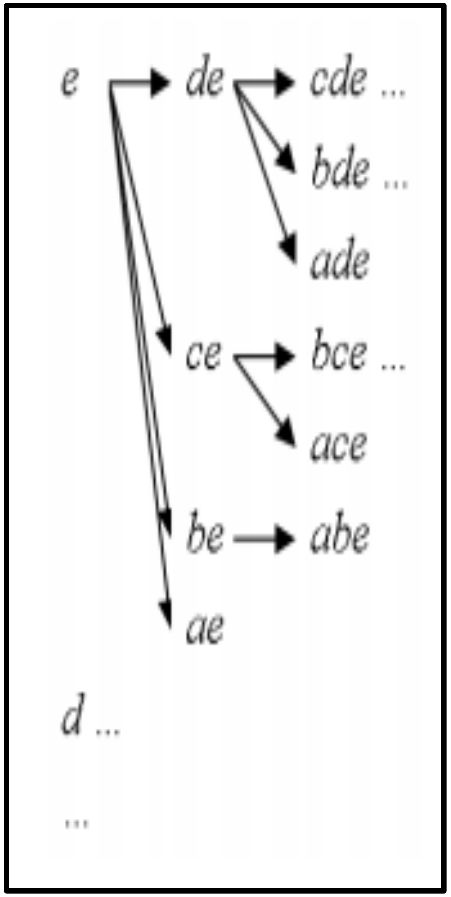
(c) Paths containing node c

(d) Paths containing node b

(e) Paths containing node a

Passo 2: Geração do *Itemset* frequente

- Cada subárvore de caminho de prefixo é processada recursivamente para extrair os *itemsets* frequentes. As soluções são então mescladas.
  - Por exemplo. a sub-árvore do caminho de prefixo para *e* será usada para extrair *itemsets* frequentes terminados em *e*, depois em *de*, *ce*, *be* e *ae*, depois em *cde*, *bde*, *cde*, etc.
  - Abordagem dividir e conquistar.

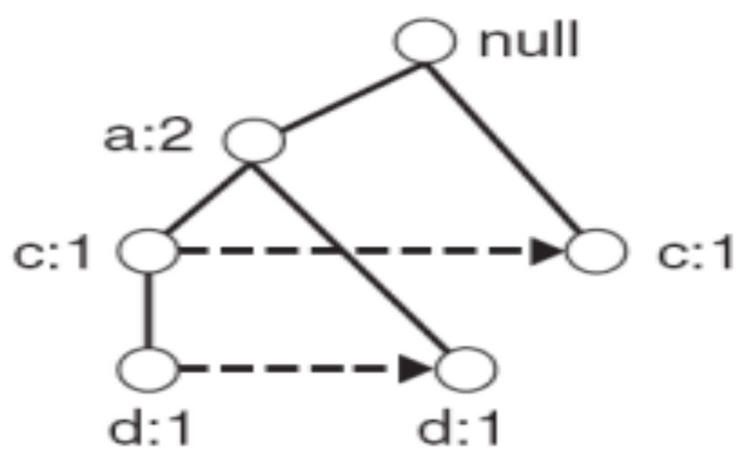


FP-Tree condicional

- A *FP-Tree* que seria construída se considerasse somente transações contendo um *itemset* particular (e então removendo esse *itemset* de todas as transações).

- Exemplo: *FP-Tree* condicional em *e*.

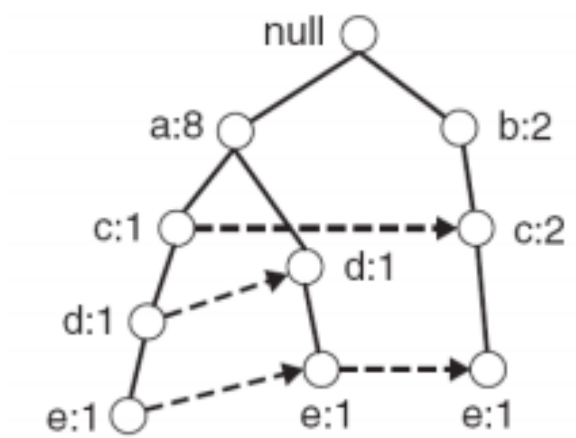
TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



Exemplo

- Seja minSup = 2 e extrai-se todos os *itemsets* frequentes contendo *e*.

- 1. Obter a sub-árvore de caminho de prefixo para *e*:



Exemplo

- 2. Verificar se *e* é um item frequente adicionando a contagem ao longo da lista ligada (linha pontilhada). Se assim, extraí-lo.
  - Sim, contagem=3, assim {*e*} é extraído como um *itemset* frequente.

- 3. Como *e* é frequente, encontrar *itemsets* frequentes terminando em *e*. i.e. *de*, *ce*, *be* e *ae*.

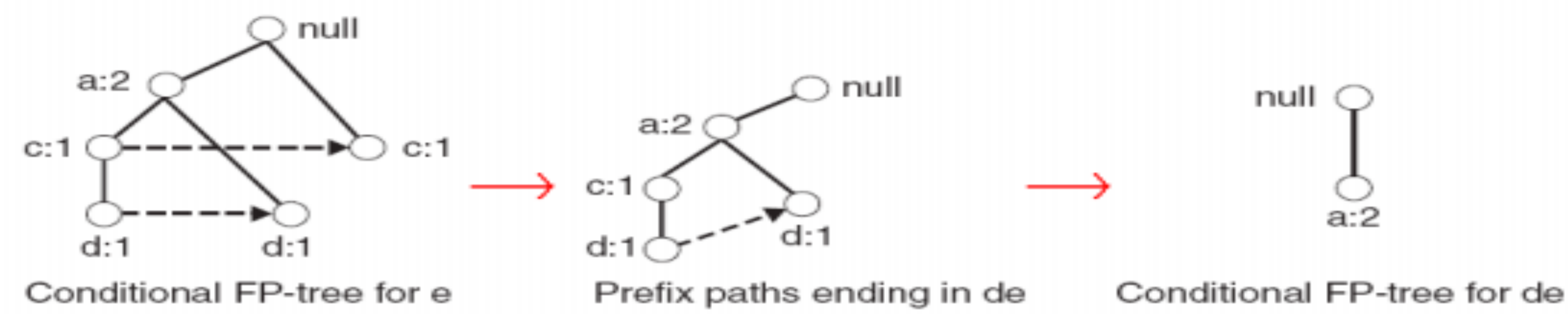
Exemplo

- 4. Usar a *FP-Tree* condicional para *e* para encontrar *itemsets* frequentes terminados em *de*, *ce* e *ae*.
  - Note que *be* não é considerado como *b* não é na *FP-Tree* condicional para *e*.

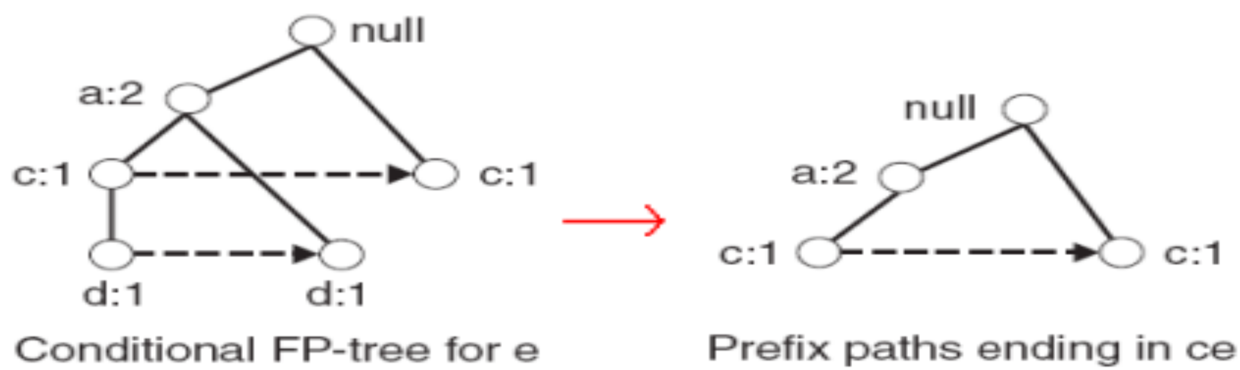
- Para cada um deles (e.g. *de*), encontrar os caminhos de prefix a partir da árvore condicional para *e*, extrair *itemsets* frequentes, gerar FP-Tree condicional, etc... (recursivo).

Exemplo

- Exemplo: *e* -> *de* -> *ade* ({*d*,*e*}, {*a*,*d*,*e*} são encontrados para ser frequentes)



- Exemplo: *e* -> *ce* ({*c*,*e*} é encontrado para ser frequente)



Resultado

Itemsets frequentes encontrados (ordenados por sufixo e na ordem em que foram encontrados):

Transaction
Data Set

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

Discussão

- Vantagens da *FP-Growth*:
  - Apenas duas passagens pelo conjunto de dados.
  - "Comprime" conjunto de dados.
  - Nenhuma geração de candidatos.
  - Muito mais rápido que o *Apriori*.

- Desvantagens da *FP-Growth*:
  - *FP-Tree* pode não caber na memória!
  - *FP-Tree* é cara para construir.

Por que o *FP-Growth* é rápido?

- Estudos de performance mostram que:
  - O *FP-Growth* é uma ordem de magnitude mais rápido que o *Apriori*.

- Motivos:
  - Não há geração ou teste de candidatos.
  - Estrutura de dados compacta.
  - Elimina iterações repetidas com o banco.

Algoritmos

- *Apriori e AprioriTid* [Agrawal R. & R. Srikant (1994)];
- *Opus* [Webb G. I. (1995)];
- *Direct Hasing and Pruning* (DHP) [Adamo J.M.(2001)];
- *Dynamic Set Couting* (DIC) [Adamo J.M. (2001)];
- *Charm* [Zaki M. & C. Hsiao (2002)];
- *FP-growth* [J. Han, J. Pei & Y. Yin (1999)];
- *Closet* [Pei J., J. Han & R. Mao (2000)];

Observações

- Diferenças entre os algoritmos:
  - forma como os dados são carregados na memória;
  - tempo de processamento;
  - tipos de atributos (numéricos, categóricos);
  - maneira com que os itemsets são gerados.

Bibliotecas e exemplos em Python

- Mlxtend
- Orange

- https://pbpython.com/market-basket-analysis.html
- https://www.kaggle.com/datatheque/association-rules-mining-market-basket-analysis
- https://orange.readthedocs.io/en/latest/reference/rst/Orange.associate.html
- https://adataanalyst.com/machine-learning/apriori-algorithm-python-3-0/
- https://github.com/calee0219/Python3-Fp-growth
- https://pypi.org/project/pyfpgrowth/